Folders: A Visual Organization System for MIT App Inventor

Xixi “Shirley” Lu

Submitted in Partial Fulfillment of the
Prerequisite for Honors in Computer Science

May 2015

Abstract

In blocks programming languages, such as MIT App Inventor, programs are built by composing puzzle-shaped fragments on a 2D workspace. Their visual nature makes programming more accessible to novices, but it also has numerous drawbacks. Users must decide where to place blocks on the workspace, and these placements may require the reorganization of other blocks. Block representations are less space efficient than their textual equivalents. Finally, the fundamental 2D nature of the blocks workspace makes it more challenging to search and navigate than the traditional linear workflow. Because of these barriers, users have difficulty creating and navigating complex programs.

In order to address these drawbacks, I have developed Folders, a visual organization system, for App Inventor. Folders, which are modeled after the hierarchical desktop metaphor folders, allow users to nest blocks within them, and solve many of the aforementioned problems. First, users can use Folders, rather than spatial closeness, to place and organize blocks, thereby explicitly indicating a relationship between them. Second, Folders allow users to selectively hide and show particular groups of blocks and address the issue of limited visible space. Lastly, users are already familiar with the folder metaphor from other applications, so their introduction does not complicate App Inventor.

Unfortunately, Folders also introduce new obstacles. Users might expect that putting blocks into Folders removes them from the main workspace semantically. However, Folders are only for organizing blocks and decluttering the workspace, and their contained blocks are still considered part of the main workspace. Furthermore, Folders exacerbate the search and navigation problem. Since blocks can now be hidden in collapsed Folders , finding a usage or declaration of a variable, procedure, or component can be more difficult. I have received preliminary feedback on my initial implementation of Folders and am designing a user study to evaluate my Folders system.

Acknowledgements

Many thanks and endless appreciation to the following people and organizations for their immeasurable support and guidance:

Professor Franklyn Turbak

Professor Eni Mustafaraj

Professor Brian Tjaden

Paul Medlock-Walton

Neil Fraser

MIT App Inventor Development Team

Wellesley Computer Science Department

Wellesley Stone-Davis Dining Hall Staff

Jonathon Andrew Beliveau, II

Angela Ai

Celia Honigberg

Jeanne Xu

Alice Liao

Wellesley Wushu Team

Wellesley Quidditch Team

Contents

Chapter 1

Introduction

1.1 Blocks Programming Languages

Blocks programming languages allow users to manipulate code fragments, which are represented as composable visual blocks that often look like jigsaw puzzle pieces. These languages are not new but more and more of them are coming into popular use today. Blocks languages are used for education (e.g. Alice[1], MIT App Inventor[2], Scratch[3]), multimedia (e.g. Wirefusion [4]), video game programming (e.g. Stencyl[5], GameBlox[6]), and more. The spatial arrangement of blocks, each representing code fragments, create computer programs. Blocks languages, such as the ones mentioned above, provide users with a library of code fragments to use in development.

These languages are frequently used in classrooms for many reasons. Blocks programming languages are easy to learn. Often, they are created for beginner programmers to lower the barrier to computing. By providing a library of code fragments by means of a graphical menu, users do not need to memorize the functions with which they create programs. Many languages, such as MIT App Inventor, come with a plug and socket metaphor suggesting how blocks can be combined and real-time error checking tools, preventing users from making certain syntax errors and compile time errors. Blocks languages also teach the concept of abstraction from the very beginning. Each of the blocks represent an abstraction of high-level behavior in their application. Lastly, learning programming with blocks create a tactile and kinesthetic learning environment in addition to the visual and auditory learning environments inherent to the classroom. This, again, makes computing more accessible because students with different learning styles are given the opportunity to step into computing.

Despite these advantages, blocks languages have many drawbacks. Blocks languages come with large 2D representations of code and their spatial manipulation is cumbersome. A number of these languages present the user with a 2D workspace, which is very different from the traditional linear workspace. The 2D nature of the workspace means programmers can only see one frame of the development workspace at a time. If a large number of block segments are needed to create a program, it would not be possible for users to view all of them at once. Lastly, sensible organization and navigation of blocks in an infinitely expanding 2D space is difficult, to say the least.

1.2 MIT App Inventor

MIT App Inventor is an online blocks programming language that allows users with little to no programming experience to create Android applications. The environment is based on the Blockly framework, a web-based graphical programming editor where users create computer programs by connecting jigsaw-puzzle-shaped blocks representing program fragments.

App Inventor was first developed at Google as an open-source web application but has since migrated to development at MIT. The current version of App Inventor is in its second iteration, developed and maintained by the App Inventor Development Team. App Inventor Classic refers to the first iteration developed at Google.

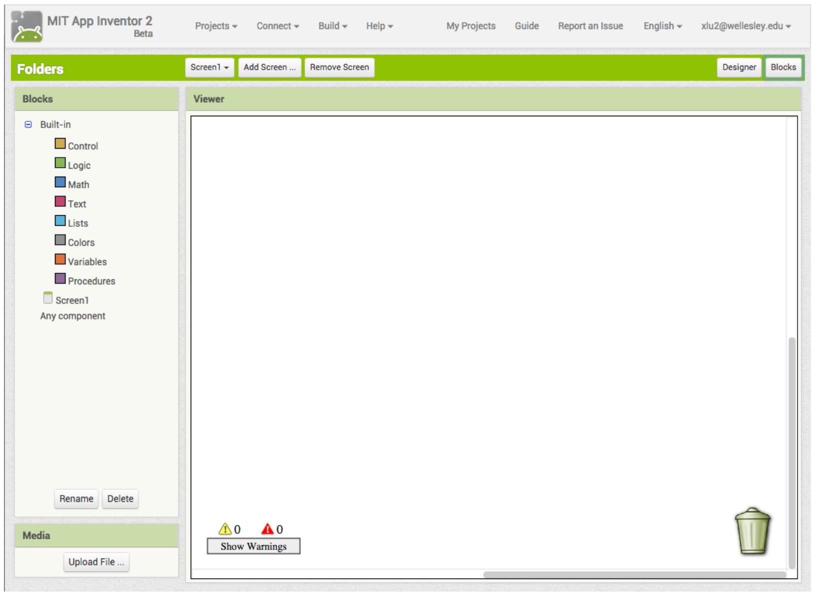
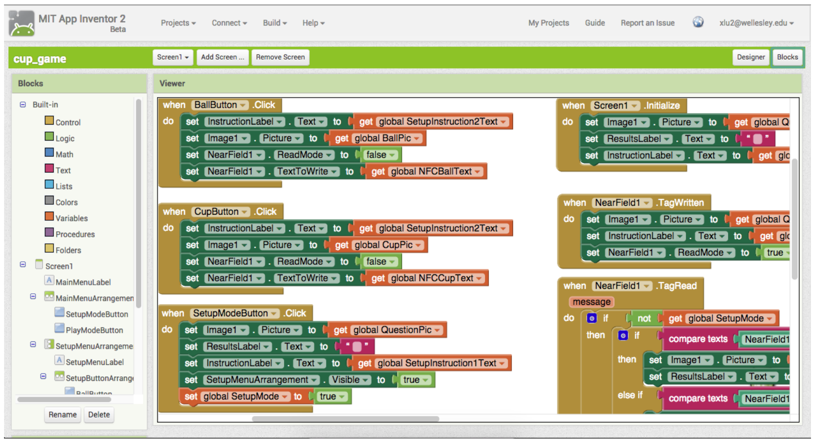


Figure 1-1: App Inventor blocks editor

1.3 Challenges Facing Blocks Programmers

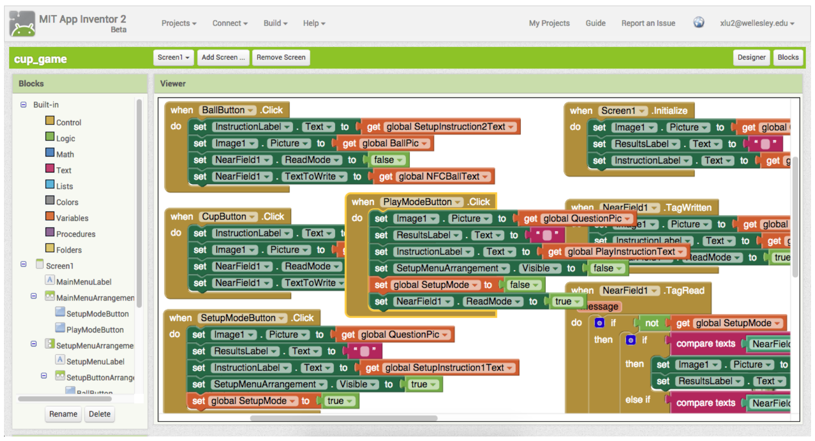
The visual nature of blocks programming makes programming more accessible to novices but also has numerous drawbacks and presents several challenges to its users. First, users must deliberately decide where to place blocks on the workspace and these placements may require the reorganization of other blocks. Second, block representations of code are less space efficient than their textual counterparts. Third, the fundamental 2D nature of the blocks workspace makes it more challenging to search and navigate than the traditional linear workflow.

Because spatial closeness is the only method for blocks organization in App Inventor, blocks programmers are forced to consistently shift block placements around in the workspace. The necessity to constantly reorganize blocks, in order to demonstrate a semblance of logical flow, is tedious. Additionally, adding new blocks becomes a very involved process. Programmers may find it preferable to use inelegant code rather than creating a new procedures and use abstraction methods.



```
when BallButton .Click
do  set InstructionLabel . Text to get global SetupInstruction2Text
    set Image1 . Picture to get global BallPic
    set NearField1 . ReadMode to false
    set NearField1 . TextToWrite to get global NFCBallText

when CupButton .Click
do  set InstructionLabel . Text to get global SetupInstruction2Text
    set Image1 . Picture to get global CupPic
    set NearField1 . ReadMode to false
    set NearField1 . TextToWrite to get global NFCCupText

when SetupModeButton .Click
do  set Image1 . Picture to get global QuestionPic
    set ResultsLabel . Text to " "
    set InstructionLabel . Text to get global SetupInstruction1Text
    set SetupMenuArrangement . Visible to true
    set global SetupMode to true

when Screen1 .Initialize
do  set Image1 . Picture to get global Q
    set ResultsLabel . Text to " "
    set InstructionLabel . Text to get glo

when NearField1 .TagWritten
do  set Image1 . Picture to get global Q
    set InstructionLabel . Text to get glo
    set NearField1 . ReadMode to true

when NearField1 .TagRead
  message
do  if not get global SetupMode
    then  if compare texts NearField1
          then  set Image1 . Picture to
                set ResultsLabel . Text to
          else if compare texts NearField1
```

Figure 1-2: A spatially organized App Inventor program



```
when BallButton .Click
do  set InstructionLabel . Text to get global SetupInstruction2Text
    set Image1 . Picture to get global BallPic
    set NearField1 . ReadMode to false
    set NearField1 . TextToWrite to get global NFCBallText

when CupButton .Click
do  set InstructionLabel . Text to
    set Image1 . Picture to
    set NearField1 . ReadMode to
    set NearField1 . TextToWrite

when PlayModeButton .Click
do  set Image1 . Picture to get global QuestionPic
    set ResultsLabel . Text to " "
    set InstructionLabel . Text to get global PlayInstructionText
    set SetupMenuArrangement . Visible to false
    set global SetupMode to false
    set NearField1 . ReadMode to true

when SetupModeButton .Click
do  set Image1 . Picture to get global QuestionPic
    set ResultsLabel . Text to " "
    set InstructionLabel . Text to get global SetupInstruction1Text
    set SetupMenuArrangement . Visible to true
    set global SetupMode to true

when Screen1 .Initialize
do  set Image1 . Picture to get global
    set ResultsLabel . Text to " "
    set InstructionLabel . Text to get g

when NearField1 .TagWritten

when NearField1 .TagRead
message
do  if not get global SetupMode
    then if compare texts NearField
         then set Image1 . Picture to
              set ResultsLabel . Text
         else if compare texts NearField
```

Figure 1-3: The same program with an additional block causing overlap and organizational discord

Since blocks are inherently more spatially involved than their textual equivalents, programmers must also deal with the fact that only a percentage of their code can be visible in one workspace window. Without a clearer overview of their code, programmers could not realize that a similar procedure had already been defined and recreate an existing procedure. On top of that, a lot of scrolling is required to survey the entire workspace (Figure 1-4). By the time the programmer reaches the block they were searching for, they could have forgotten the reason they were trying to find the block in the first place.

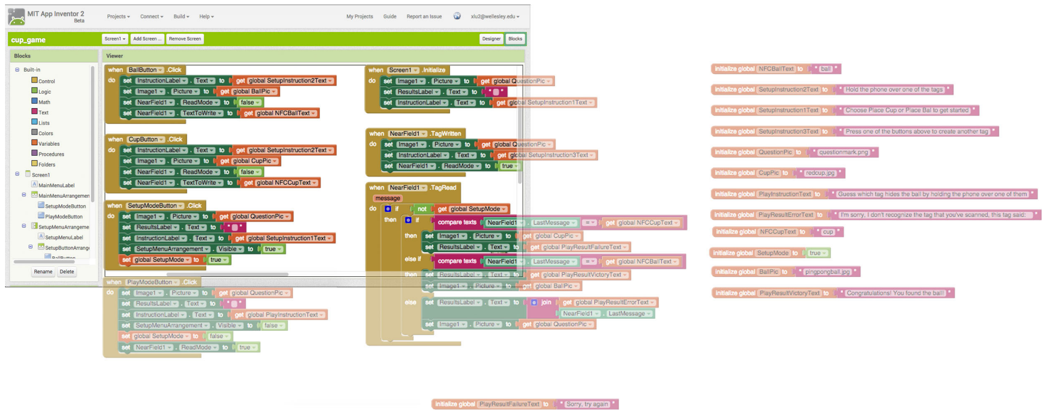


Figure 1-4: Scrolling is necessary to see all the blocks in the workspace

Searching and navigating the workspace is, unfortunately, a problem inherent to all blocks programming languages with a 2D workspace. With textual languages, tools have been developed for searching and navigation within linear workspaces. Text editors associated with programming, such as Sublime, IntelliJ, and Emacs create the workspace for the, traditionally vertical, linear workflow of textual languages. While users can build programs horizontally, it is considered best practice to limit the number of characters on each line. In fact, there are coding standards associated with each textual programming language; however, the lack of coding standards for blocks programming languages is glaring.

In order for programmers to utilize blocks programming languages to their maximum potential, a number of tools must be developed. These tools include searching and navigation, blocks organization, and more. This thesis focuses on the design and development of a blocks organization tool. The tool should, ideally, be recognizable and easy for users to pick up. After this tool is developed, coding standards for blocks programming languages can be developed and implemented using the new system for blocks organization.

1.4 Existing Tools for Visual Organization in App Inventor

As mentioned previously, App Inventor programs often require large numbers of blocks. In fact, applications with Google Play Store marketable value would require hundreds of blocks. While App Inventor does provide some methods of organizing blocks, users cannot easily organize the blocks exactly the way they want. These block organization tools are available to the user in a right-click drop down menu (Figure 1-5). The tools are split into two varieties – view and arrange.

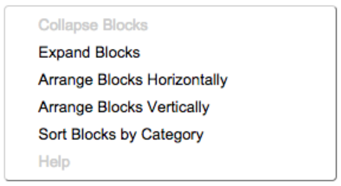


Figure 1-5: Block segment view and arrange options available through right-click drop down menu

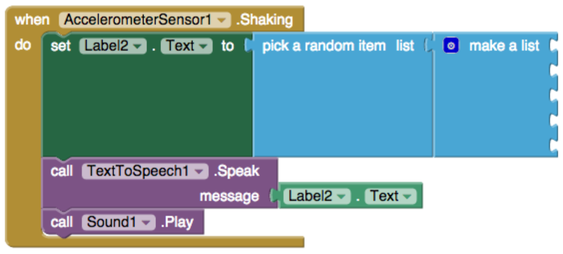

when AccelerometerSensor1 .Shaking
do set Label2 . Text to pick a random item list make a list
call TextToSpeech1 .Speak
message Label2 . Text
call Sound1 .Play

Figure 1-6: Expanded block segment

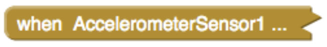


Figure 1-7: Collapsed block segment

There are two ways of viewing block segments — expanded (Figure 1-6) and collapsed (Figure 1-7); and 3 ways of arranging block segments — horizontally (Figure 1-8 and 1-11), vertically (Figure 1-9 and 1-12), and by category (Figure 1-10 and 1-13). From Figures 1-8- 1-13, the differences between each of the view and arrange options can be seen.

While these options are useful for spreading blocks out onto the development workspace and allowing users to see each block segment, users must prioritize access to the individual blocks or visualization of the entire program. Although in smaller programs this becomes less of a problem, programs with "real use" are rarely small enough that every block can be seen without scrolling through the workspace. Block organization is also less of a problem in smaller programs because users can see every block in one frame of the development workspace.

In addition to the prioritization dilemma, there is also no simple way for App Inventor to take user preference into organizing block segments beyond horizontal, vertical, and category. If a user placed Block Segment 1 and Block Segment 2 close to each other on the development workspace before using one of the arrange options, there is no guarantee Block Segment 1 and 2 will remain close to each other. Furthermore, collapsing a block makes it difficult to understand the purpose and structure of the program and can create visibility problems when expanded. Since users tend to have preferences in the way they organize code, these right-click options, while powerful, have numerous drawbacks.

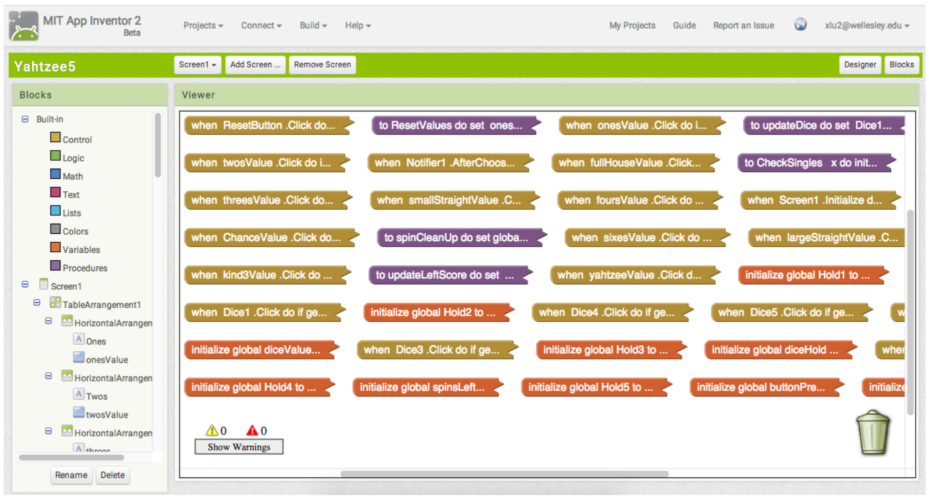

MIT App Inventor 2 Beta
Projects Connect Build Help
My Projects Guide Report an Issue xlu2@wellesley.edu
Yahtzee5
Screen1 Add Screen ... Remove Screen
Designer Blocks
Blocks
Viewer
Built-in
Control
Logic
Math
Text
Lists
Colors
Variables
Procedures
Screen1
TableArrangement1
HorizontalArrangen
Ones
onesValue
Twos
twosValue
when ResetButton .Click do...
to ResetValues do set ones...
when onesValue .Click do i...
to updateDice do set Dice1...
when twosValue .Click do i...
when Notifier1 .AfterChoos...
when fullHouseValue .Click...
to CheckSingles x do init...
when threesValue .Click do...
when smallStraightValue .C...
when foursValue .Click do ...
when Screen1 .Initialize d...
when ChanceValue .Click do...
to spinCleanUp do set globa...
when sixesValue .Click do ...
when largeStraightValue .C...
when kind3Value .Click do ...
to updateLeftScore do set ...
when yahtzeeValue .Click d...
initialize global Hold1 to ...
when Dice1 .Click do if ge...
initialize global Hold2 to ...
when Dice4 .Click do if ge...
when Dice5 .Click do if ge...
initialize global diceValue...
when Dice3 .Click do if ge...
initialize global Hold3 to ...
initialize global diceHold ...
initialize global Hold4 to ...
initialize global spinsLeft...
initialize global Hold5 to ...
initialize global buttonPre...
0 0
Show Warnings
Rename Delete

Figure 1-8: Collapsed block segments organized horizontally

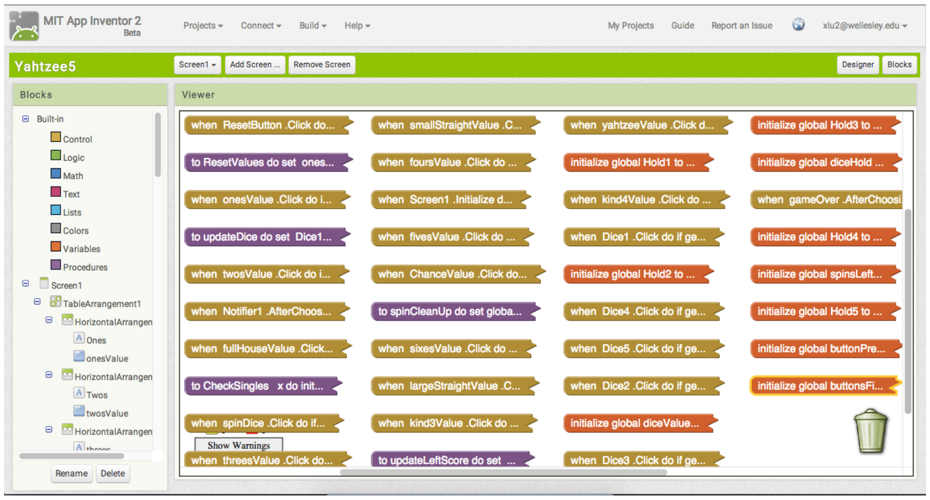
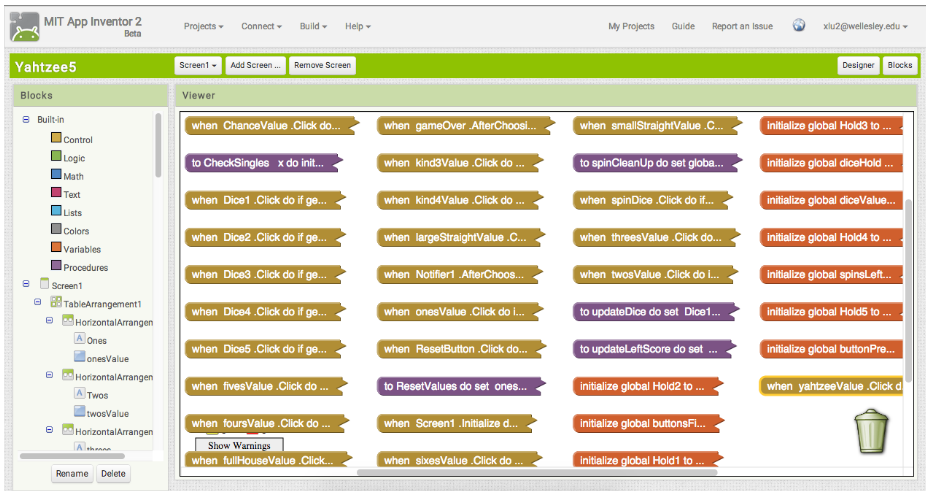
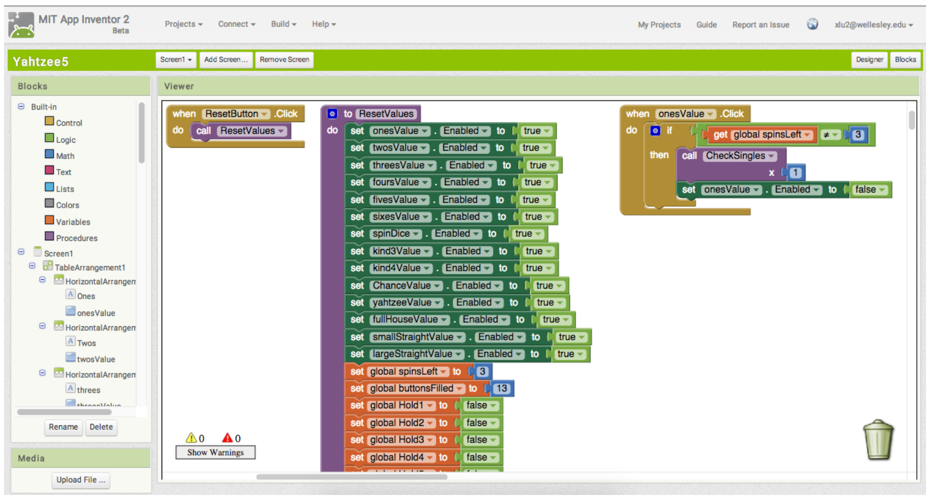

MIT App Inventor 2 Beta
Projects Connect Build Help
My Projects Guide Report an Issue xlu2@wellesley.edu
Yahtzee5
Screen1 Add Screen ... Remove Screen
Designer Blocks
Blocks
Viewer
when ResetButton .Click do...
when smallStraightValue .C...
when yahtzeeValue .Click d...
initialize global Hold3 to ...
to ResetValues do set ones...
when foursValue .Click do ...
initialize global Hold1 to ...
initialize global diceHold ...
when onesValue .Click do i...
when Screen1 .Initialize d...
when kind4Value .Click do ...
when gameOver .AfterChoosi
to updateDice do set Dice1...
when fivesValue .Click do ...
when Dice1 .Click do if ge...
initialize global Hold4 to ...
when twosValue .Click do i...
when ChanceValue .Click do...
initialize global Hold2 to ...
initialize global spinsLeft...
when Notifier1 .AfterChoos...
to spinCleanUp do set globa...
when Dice4 .Click do if ge...
initialize global Hold5 to ...
when fullHouseValue .Click...
when sixesValue .Click do ...
when Dice5 .Click do if ge...
initialize global buttonPre...
to CheckSingles x do init...
when largeStraightValue .C...
when Dice2 .Click do if ge...
initialize global buttonsFi...
when spinDice .Click do if...
when kind3Value .Click do ...
initialize global diceValue...
when threesValue .Click do...
to updateLeftScore do set ...
when Dice3 .Click do if ge...
Show Warnings
Rename Delete

Figure 1-9: Collapsed block segments organized vertically



```
when ChanceValue .Click do...
to CheckSingles x do init...
when Dice1 .Click do if ge...
when Dice2 .Click do if ge...
when Dice3 .Click do if ge...
when Dice4 .Click do if ge...
when Dice5 .Click do if ge...
when fivesValue .Click do ...
when foursValue .Click do ...
when fullHouseValue .Click...
when gameOver .AfterChoosi...
when kind3Value .Click do ...
when kind4Value .Click do ...
when largeStraightValue .C...
when Notifier1 .AfterChoos...
when onesValue .Click do i...
when ResetButton .Click do...
to ResetValues do set ones...
when Screen1 .Initialize d...
when sixesValue .Click do ...
when smallStraightValue .C...
to spinCleanUp do set globa...
when spinDice .Click do if...
when threesValue .Click do...
when twosValue .Click do i...
to updateDice do set Dice1...
to updateLeftScore do set ...
initialize global Hold2 to ...
initialize global buttonsFi...
initialize global Hold1 to ...
initialize global Hold3 to ...
initialize global diceHold ...
initialize global diceValue...
initialize global Hold4 to ...
initialize global spinsLeft...
initialize global Hold5 to ...
initialize global buttonPre...
when yahtzeeValue .Click d
```

Figure 1-10: Collapsed block segments organized by category



```
when ResetButton .Click
do  call ResetValues

to ResetValues
do  set onesValue . Enabled to true
    set twosValue . Enabled to true
    set threesValue . Enabled to true
    set foursValue . Enabled to true
    set fivesValue . Enabled to true
    set sixesValue . Enabled to true
    set spinDice . Enabled to true
    set kind3Value . Enabled to true
    set kind4Value . Enabled to true
    set ChanceValue . Enabled to true
    set yahtzeeValue . Enabled to true
    set fullHouseValue . Enabled to true
    set smallStraightValue . Enabled to true
    set largeStraightValue . Enabled to true
    set global spinsLeft to 3
    set global buttonsFilled to 13
    set global Hold1 to false
    set global Hold2 to false
    set global Hold3 to false
    set global Hold4 to false

when onesValue .Click
do  if  get global spinsLeft ≠ 3
    then call CheckSingles
              x 1
         set onesValue . Enabled to false
```

Figure 1-11: Expanded block segments organized horizontally

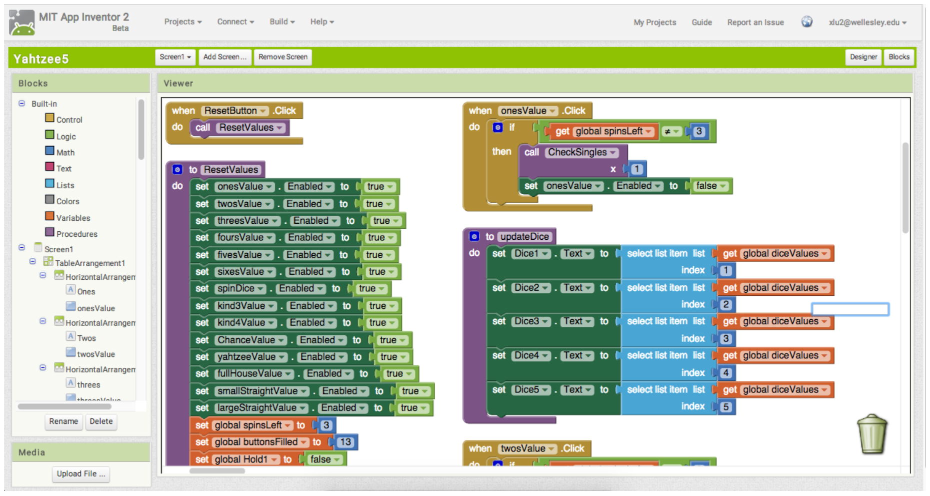


Figure 1-12: Expanded block segments organized vertically

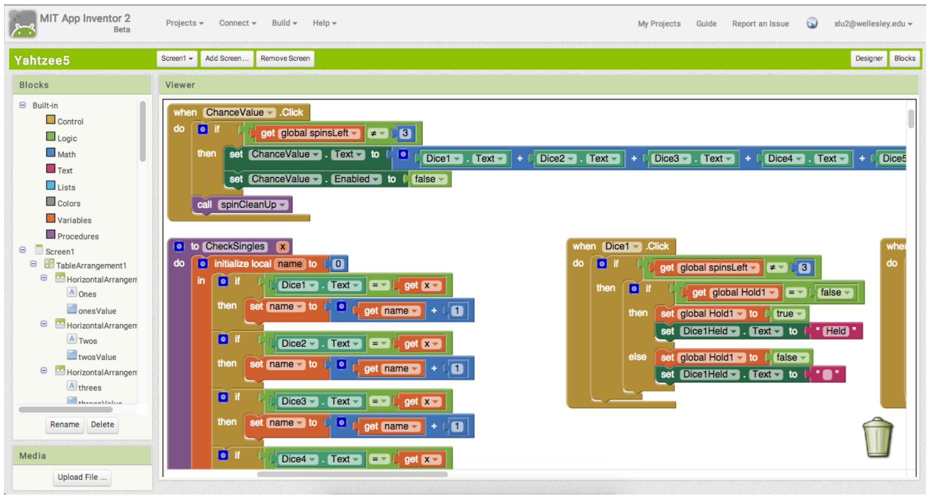


Figure 1-13: Expanded block segments organized by category

1.5 Folders : A New Blocks Organization Tool

In order to address the deficiency in user preference-driven block segment organization, I am introducing Folders , a visual organizational tool for App Inventor.

1.5.1 Main Workspace

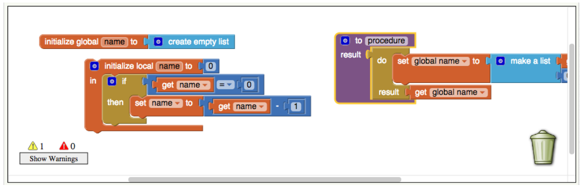


Figure 1-14: The main workspace

The main workspace is the development surface in App Inventor's blocks editor. There is a warning and errors indicator, trashcan, and user created blocks on the main workspace.

1.5.2 Mini-Workspaces

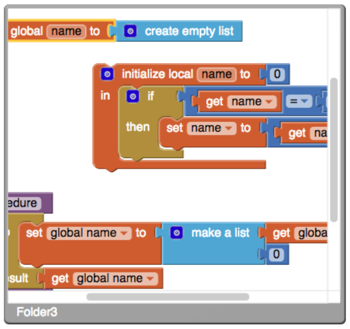


Figure 1-15: A mini-workspace

I developed mini-workspaces (Figure 1-15) for FOLDERS . Mini-workspaces are, as the name implies, smaller workspace instances. A mini-workspace has the same properties and functionality (e.g. scrollable, contains blocks, etc) as the main workspace but is a member of the main workspace (Figure 1-14). Multiple mini-workspaces can be expanded and can overlap. Mini-workspaces do not have a separate warning and error indicators or trashcan.

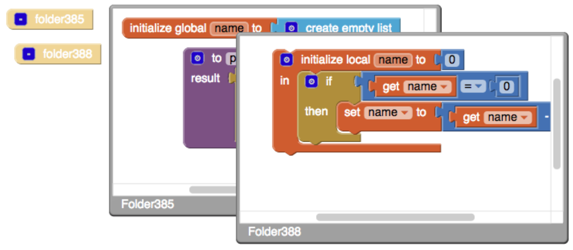


Figure 1-16: Overlapping mini-workspaces

1.5.3 What are Folders ?

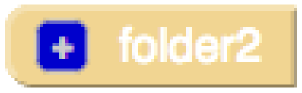


Figure 1-17: A Folder

A Folder can exist in two states - expanded and minimized. A minimized Folder is the Folder block shown in Figure 1-17. An expanded Folder consists of two parts - the Folder block and the Folder's mini-workspace, shown in Figure 1-18. Blocks in the Folder are shown in the mini-workspace, which is a smaller development workspace.

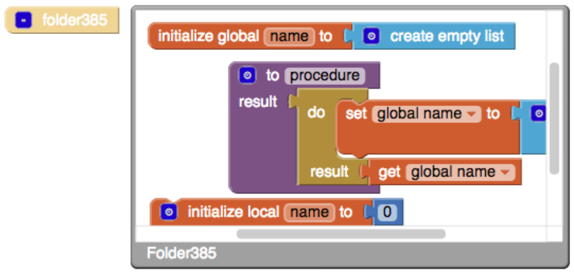


Figure 1-18: An Expanded Folder

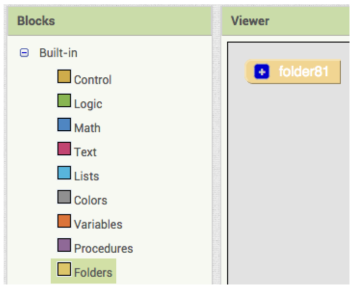


Figure 1-19: An empty Folder in the block palette

Folders are created by pulling a minimized, empty Folder out of the block palette, shown in Figure 1-19. Block segments can be added or removed from Folders by 1) expanding a Folder and 2) dragging the block into or out of the Folder's mini-workspace.

For more details on the design and usage of Folders, including numerous functions not described here, see Section 3.1 Design.

1.5.4 Design of Folders

The driving force behind the design of Folders is the need for a user-preference driven blocks organization tool. Folders should not interfere with the existing methods for block rearrangement (arrange horizontally, arrange vertically, sort blocks by category) and block display (collapsed, expanded), instead provide a new set of functionality.

Many of the design choices made for Folders were for the sake of familiarity to the user. When possible, the appearance and behavior of Folders in App Inventor perfectly mimic the behavior of folders on Desktop computers. The addition of Folders to App Inventor is not meant to further complicate App Inventor, but rather simplify. Thus, by using the design of Desktop folders as the foundation of App Inventor Folders, users will already have some familiarity with its usage and behavior.

Presenting users with a tool they are already so familiar with does create complications. If Folders for App Inventor does not replicate the expected behavior correctly, there will be significant dissonance for the user. This may cause the user to become frustrated with App Inventor as a whole and discourage he or she from continuing to use App Inventor. However, the necessity for a tool such as Folders far outweighs its potential drawbacks.

1.5.5 Why Folders ?

Users are already familiar with the Desktop metaphor and the paper paradigm, with it the idea of using virtual folders to organize files, images, and other folders. In a 2002 study on the design of computer icons, it was shown that a user's familiarity with the icon is very important [7]. The more recognizable an icon, the more locatable and usable. Since users are familiar with folders, the usability and simplicity of Folders will help users adapt to using them while creating Android applications in App Inventor.

In addition, the familiarity of Folders does not result in further steepening of the App Inventor learning curve. App Inventor is a teaching tool with a significant learning curve. Quite often, first-time App Inventor users are also young, beginner coders. Teaching tools are most effective if it doesn't take long to learn how to use the tools itself. Through user studies, I hope to show that Folders do not make App Inventor more difficult and can improve the user experience for first-time users and seasoned users.

There is one distinction between App Inventor Folders and Desktop metaphor folders - although block segments are "in" Folders , they are only visually organized "into" Folders . The block segments are still in the main workspace, the block segments are still enumerated as in the main workspace, the parent blocks are still considered top blocks of the main workspace, and the Folder block isn't considered a top block of the main workspace. In other words, there are no semantic changes with the introduction of Folders .

Folders give the user a lot of freedom in terms of organizing blocks. The way blocks are or can be organized is entirely up to the user and their preference. Users can place relevant block segments into separate Folders rather than different parts of the development workspace. This will also allow users to take advantage of the right-click arrange options.

Lastly, by placing block segments into Folders and minimizing them, users can now work on one block segment without running into another block segment. The development workspace in blocks programming languages are limiting and developing in such a workspace can result in blocks connecting with the wrong blocks or users running out of space to develop in. Folders will truly allow users to put blocks away and let users develop without worrying about running out of space or creating unintentional connections (Figures 1-20 and 1-21).

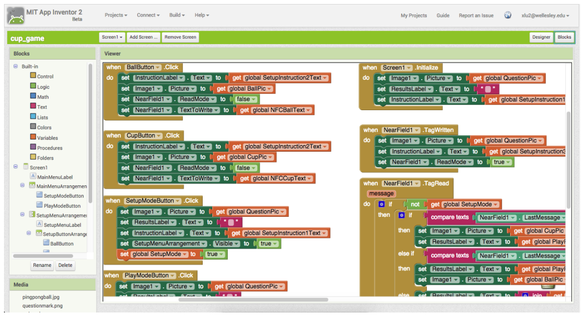


Figure 1-20: A large program without Folders

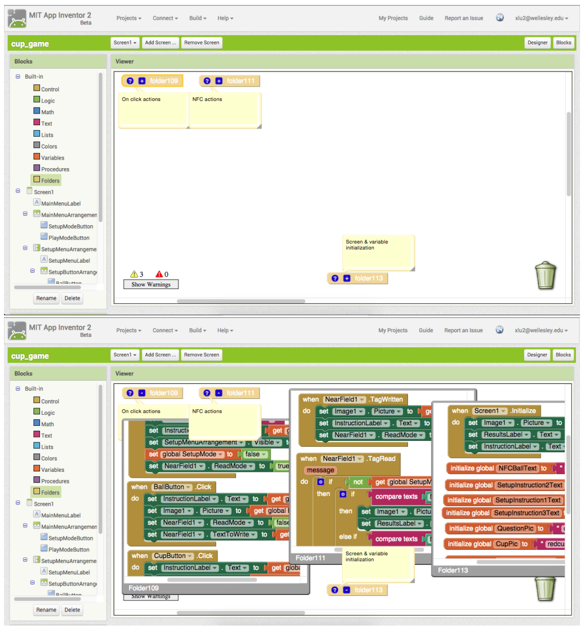


Figure 1-21: The same program with Folders

1.6 Development Progress & Hurdles

Development for Folders happened in 3 distinct stages: 1) Designing and prototyping the appearance and behavior of Folders , 2) Developing the visual appearances of Folders , and 3) Implementing the behavior of Folders . In each of these stages, the scalable vector graphics (SVG) nature of Folders and App Inventor's blocks editor presented itself in numerous challenges.

Blocks programming languages are inherently difficult to develop for - the correct and expected behavior is neither simple or easy to create. For example, blocks and Folders can only exist in one workspace and that workspace is bounded. When a block or Folder is dragged out of a Folder's workspace, it should be entirely visible as it crosses a boundary. To implement this correct behavior, every time a mouseDown action is captured for a block or Folder, the block or Folder is moved to the larger workspace. On mouseUp, if the block or Folder is over a mini workspace, it will be moved into the mini workspace. Each of these challenges will be discussed thoroughly in Chapter 4 - Implementation.

1.7 Road Map

The rest of this document is organized as below:

- Chapter 2 - Related Work discusses my past blocks editor work with App Inventor, folders of the Desktop Metaphor, and how other blocks programming languages tackle the blocks organization problem.
- Chapter 3 - Design explores the intentions and process of designing FOLDERS.
- Chapter 4 - Implementation details the development process of creating FOLDERS and getting FOLDERS to where it is today.
- Chapter 5 - User Studies analyzes the initial feedback and informal user testing FOLDERS received and proposes a user study for further and more thorough testing of FOLDERS.
- Chapter 6 - Conclusion and Future Work examines the current state of FOLDERS and suggests the next steps to further improve FOLDERS.
- Appendices include any code created or modified for the implementation of FOLDERS.

Chapter 2

Related Work

2.1 Past Work with MIT App Inventor

I have collaborated with the MIT App Inventor Development team on extending App Inventor's user interface, specifically with the blocks editor (Figure 1-1). Some of my previous work included zooming and scaling for App Inventor's blocks editor and creating an indented socket shape that emphasizes nesting of subexpressions.

2.1.1 Indented Value Inputs

Prior to my work with indented value inputs, App Inventor had only 2 input types - value and statement. This presented a problem with procedure and variable blocks (Figure 2-1 and 2-2). First, value inputs did not demonstrate scope of the variables properly. There is a significant visual difference between the value input and statement input versions of the same block. Second, while the statement input versions clearly showed scope of the variables initialized for the procedures, the value input versions did not. Lastly, there is a lot of space wasted with the input value versions of the block (Figure 2-3), which greatly exacerbates the problem of the already limited visibility of the workspace for programmers.

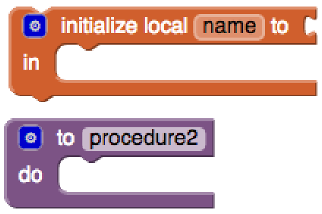


Figure 2-1: Statement input of variables & procedures

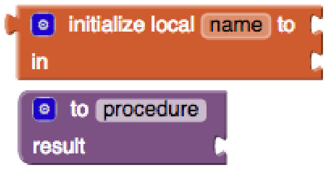


Figure 2-2: Value input of variables & procedures

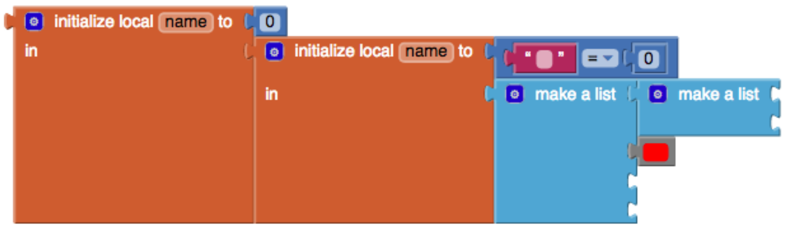


Figure 2-3: Space wasted with value input of variables & procedures

In order to solve these problems, I implemented a new input - indented value input. This new input would act as a value input in all aspects except for its appearance. Its appearance would be a hybrid of the statement input and value input. This solved all 3 of the problems presented - scope is demonstrated properly, the blocks are now visually similar, and very little space is wasted (Figure 2-5). This improvement makes a big difference because 2D representations already occupy more space than its textual counterparts.

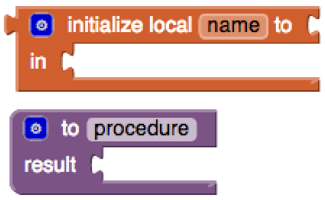


Figure 2-4: New indented value input of variables & procedures

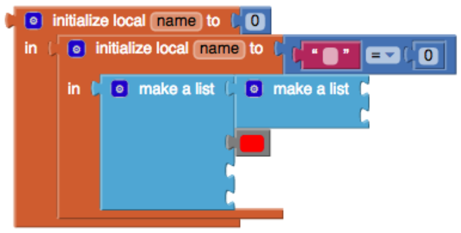


Figure 2-5: Figure 2-3 with new indented value inputs

2.1.2 Zooming & Scaling

Zooming and scaling for App Inventor's blocks workspace is another project that would greatly benefit the user interface of the Blocks Editor. In App Inventor Classic, there was a way for the user to see a mini-map of the entire workspace (Figure 2-6). Because this feature is no longer available in App Inventor, users cannot see an overview of their entire workspace. Since the visible area of the workspace is limited, zooming and scaling would allow for programmers to see more, or less, of the entire workspace.

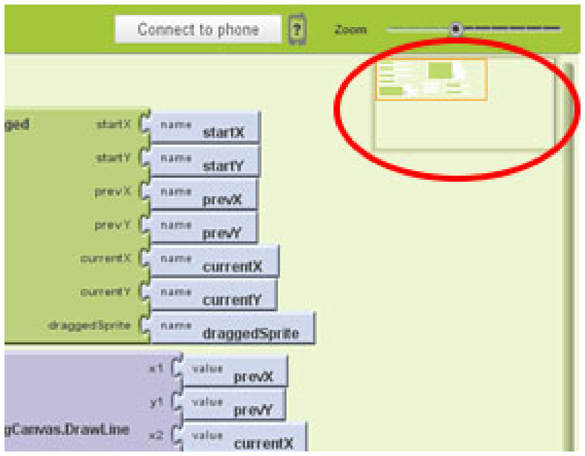


Figure 2-6: App Inventor Classic mini-map

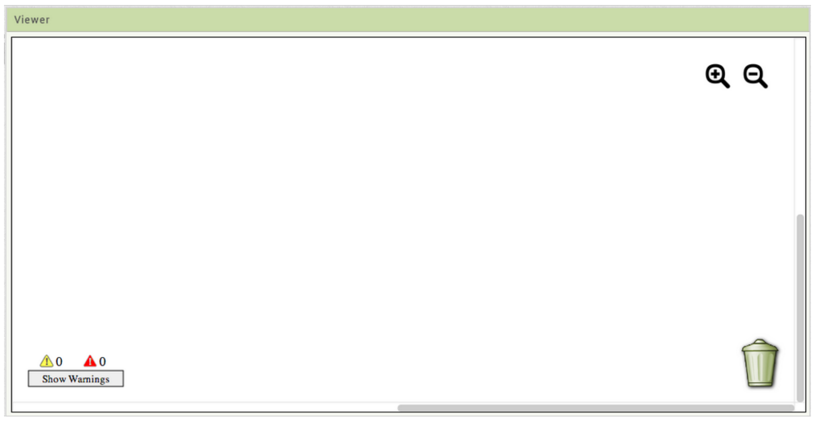


Figure 2-7: Zoom for App Inventor blocks editor

Zooming and scaling would benefit a variety of App Inventor programmers. First, users can zoom out and make more of the workspace visible. This would allow the users to have a better sense of their project and also make navigation of the workspace easier. Second, users with low vision can zoom in and better see the details of each block. Third, developers for App Inventor, especially those interested in working with the scalable vector graphics (SVG) layer of development, can see the specifics of each svg group and line, allowing for small detailed changes to the blocks editor.

I implemented zooming and scaling by creating two icons, a plus magnifying glass and a minus magnifying glass, which can be seen in the upper right hand corner of the mini-workspace. By clicking on a magnifying glass, the workspace will magnify or reduce.

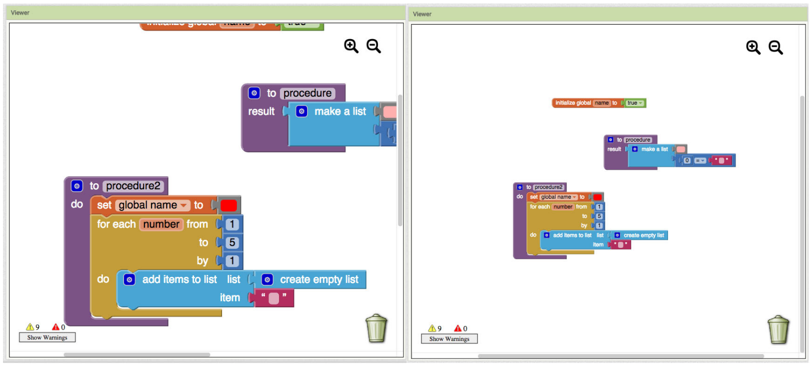


Figure 2-8: Workspace without and with Zoom

2.2 Folders of the Desktop Metaphor

The Desktop Metaphor is an unifying interface metaphor describing the graphical user interface of modern personal computers. The idea behind the Desktop Metaphor is that users can treat the computer's desktop as a physical desktop where objects such as files and folders can be placed. These objects can be opened in a window, similar to how a file or folder can be placed on top a desktop.

In this thesis, FOLDERS for App Inventor will be compared to and based off of folders of Mac OS X (Yosemite), which is an implementation of the Desktop Metaphor. The purpose and behaviors of folders I hope to implement for FOLDERS include:

- Usage of FOLDERS to "create" more space (Figure 2-9).
- Using FOLDERS as an organization method for blocks (Figure 2-10).
- Recursively nestable FOLDERS (Figure 2-11.
- Highlighting of FOLDERS to make location of blocks unambiguous (Figure 2-12).

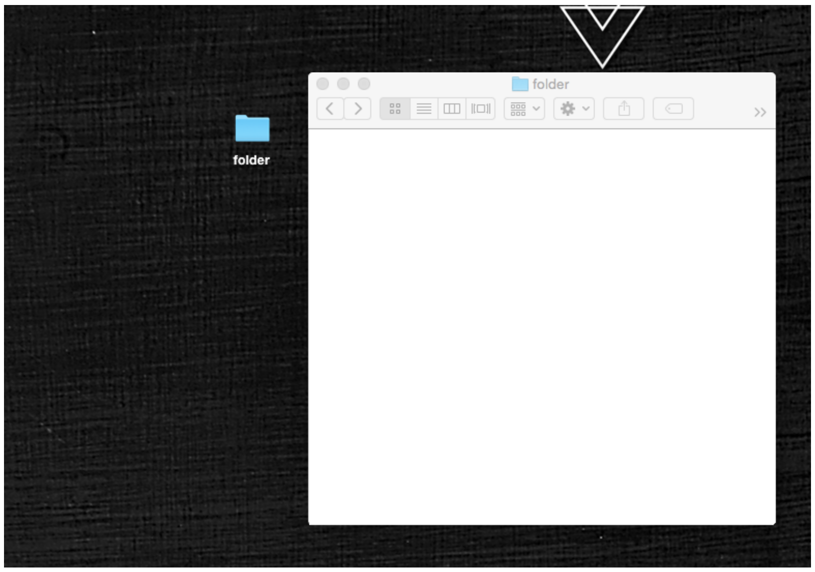


Figure 2-9: Desktop folders are used to create more space to put files, images, and other objects away.

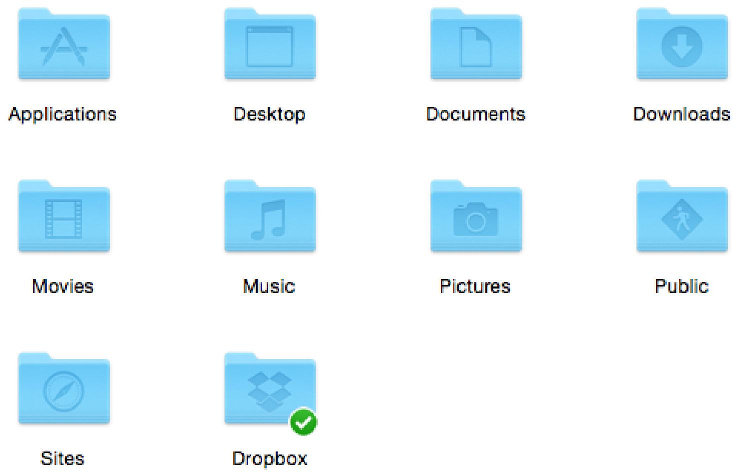


Figure 2-10: Desktop folders can be used to organize objects by functionality, type, or any other user defined method.

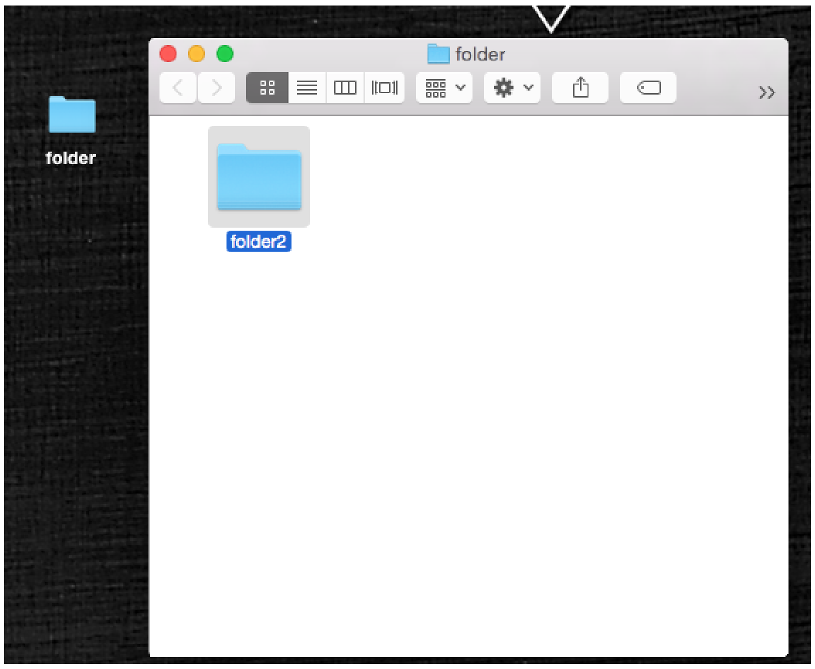


Figure 2-11: Desktop folders are nestable.

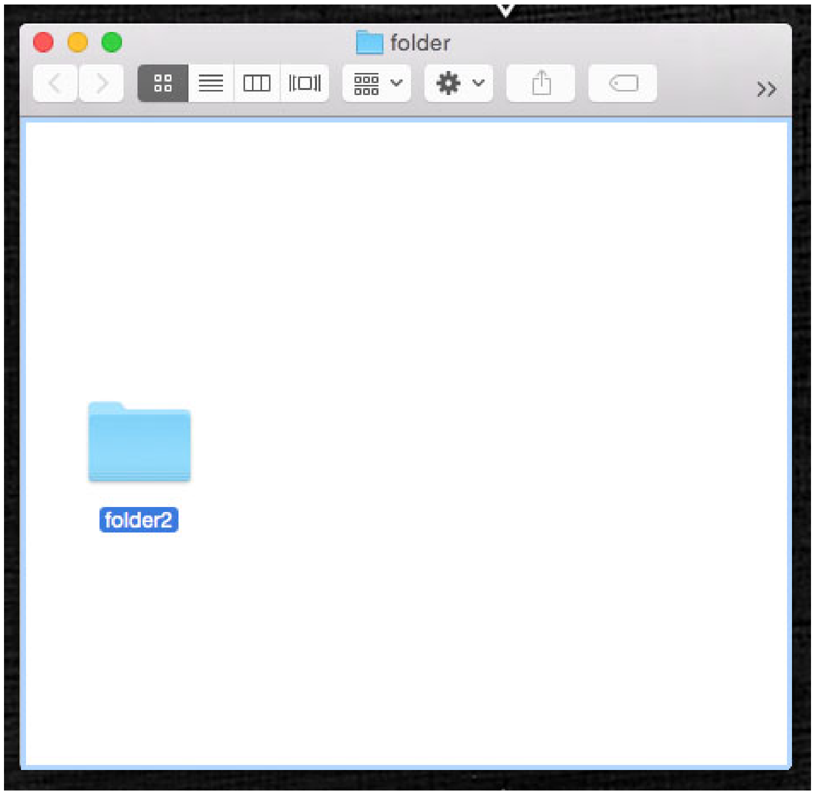


Figure 2-12: Desktop folders use highlighting to make location of objects unambiguous.

2.3 Organization

We use organization in many aspects of our digital life, such as organizing files on our computer with folders, emails in our inbox with labels, and code in our programs with files. In each of these systems, search functionalities are available in addition to the organization ones mentioned above. However, in a study conducted about search versus organization with the usage of folders in non-specific project work, participants said that it would not be possible for them to depend exclusively on search. These participants cited 3 reasons: 1) trust, 2) control, and 3) visibility/understandability [8]. Although search is becoming more prevalent nowadays, it has not been able to completely replace the necessity of organization tool likes folders. As such, it is increasingly more important to have tools like search and organization available for App Inventor users.

In terms of organization in textual programming languages, programmers have a number of tools at their disposal. With large software systems, engineers often use clustering to group procedures and variables into classes. These clusters can then be further grouped to create subsystems of behaviors [9]. Code conventions also exist for many textual programming languages. These guidelines help pass code on from one engineer to another by creating more legible and stable programs with fewer bugs [10]. Additionally, there are a number of design patterns available for different languages. The module pattern, used frequently in Javascript, is an infrastructure that maintains legible and organized source code [11].

The importance of source code organization comes from the fact that good code comes from good organization of code [12]. There are many guidelines, books, and articles on how to write good code available for programmers. The question for App Inventor, especially as a teaching tool, is how can one expect to learn how to write good code if there isn't a code organization tool in place and available. Because users still correlate source code organization with a file tree [13], creating a folder system for block organization in App Inventor is a good first stepping stone.

In the end, it is important for an organization and search system to work hand in hand, creating a good development environment. When thinking about information retrieval in computer systems, a Library-Librarian metaphor could be used. A well organized library, or source code, allows a librarian, a search tool, to find the targeted information easily [14]. With this in mind, having a good organization system is of utmost importance. Only after doing so will a search and navigation system complete the Library-Librarian metaphor for App Inventor.

2.4 Blocks Organization in Other Blocks Languages

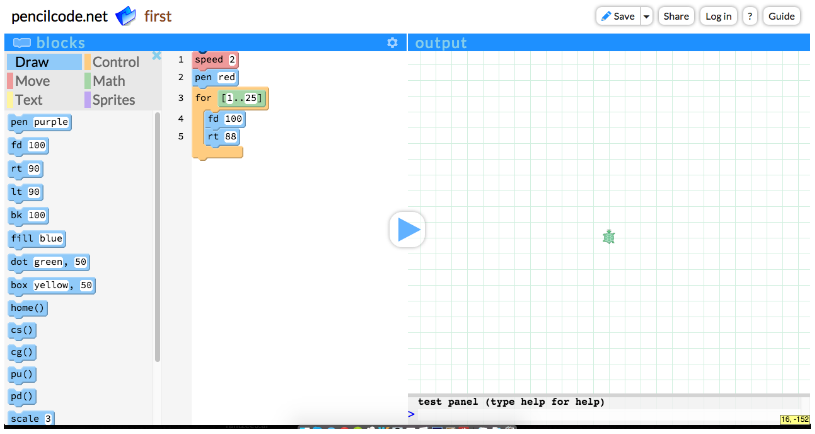
Blocks organization methods in five other blocks languages, spanning the fields of education, gaming, and multimedia, were studied to determine what efforts have been made in the field of visual programming for organization. These five languages were Pencil Code[15], Scratch[3], Stencyl[5], Gameblox[6], and WireFusion[4].

Out of these five languages, Scratch, Stencyl, and Gameblox have similar blocks organization methods and two do not. The blocks organization methods presented by those three languages followed a different design from Folders . Users can create additional workspaces but these workspaces cannot nest recursively and, with the exception of Gameblox, blocks cannot be transported between workspaces. While Folders followed a tree structure, these methods were list structured. Explicit block organization methods are not seen in any of these five languages and users must use spatial closeness to represent relationships between blocks.

2.4.1 Pencil Code

Pencil Code is a browser-based, "collaborative programming site for drawing art, playing music, and creating games. It is also a place to experiment with mathematical functions, geometry, graphing, webpages, simulations, and algorithms" [15]. Pencil Code has a linear coding environment similar to that of textual languages and code can easily be translated between blocks and text. In fact, blocks in Pencil Code are CSS stylizations of text rather than an abstraction of high-level actions.

In terms of blocks organization, Pencil Code does not allow users to create additional workspaces. In fact, the workspace is not 2D and executable code blocks must all be connected vertically. One of the advantages of the linear workspace is that Pencil Code can easily handle programs with tens of thousands of lines of code.

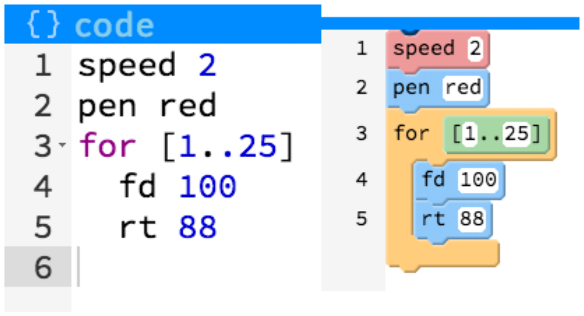


```
pen purple
fd 100
rt 90
lt 90
bk 100
fill blue
dot green, 50
box yellow, 50
home()
cs()
cg()
pu()
pd()
scale 3
```

```
speed 2
pen red
for [1..25]
  fd 100
  rt 88
```

```
test panel (type help for help)
>
```

Figure 2-13: Pencil Code user interface



```
speed 2
pen red
for [1..25]
  fd 100
  rt 88
```

Figure 2-14: Pencil Code blocks are graphical borders surrounding textual code

2.4.2 Scratch

Scratch “is a free educational programming language that was developed by the Lifelong Kindergarten Group at the Massachusetts Institute of Technology (MIT) with over 5 million registered users... Users program in Scratch by dragging blocks from the block palette and attaching them to other blocks like a jigsaw puzzle” [16].

Blocks organization in Scratch is centered around sprites, or objects that perform actions in Scratch programs. Each sprite has its own 2D scripts area. Blocks in the script areas can be cleaned up using a built-in right-click menu option. Nested workspaces and explicit blocks organization methods are not available in Scratch.

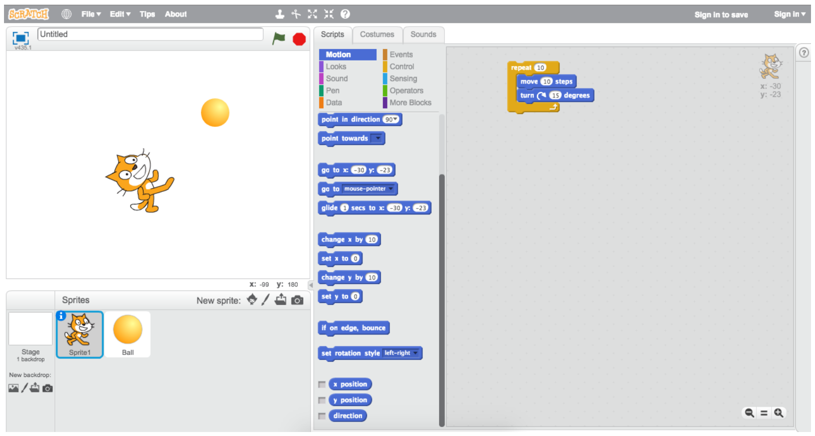


Figure 2-15: Scratch user interface

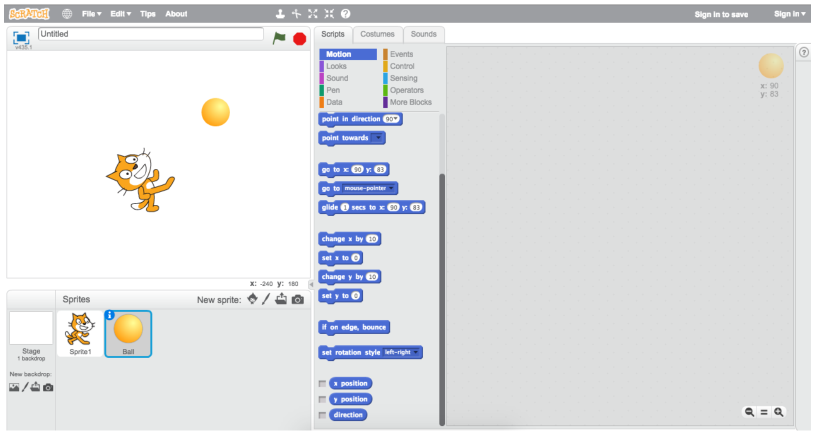


Figure 2-16: Each sprite in Scratch has its own scripts area

Figure 2-17: Users can clean up the scripts area using a right-click function

2.4.3 Stencyl

Stencyl is a mobile, web, and platform game creation tool and allows users to create these games without code. Although Stencyl is presented as a tool for creating games without code, there are options for users to program in Java or other languages as well as a conversation method between blocks and text.

In Stencyl, code is organized into behaviors, which are split into actors and scenes. Each of these behaviors have their own development workspace. Stencyl's design environment is based off of Scratch and, like Scratch, do not provide nested environments or explicit block organization methods.

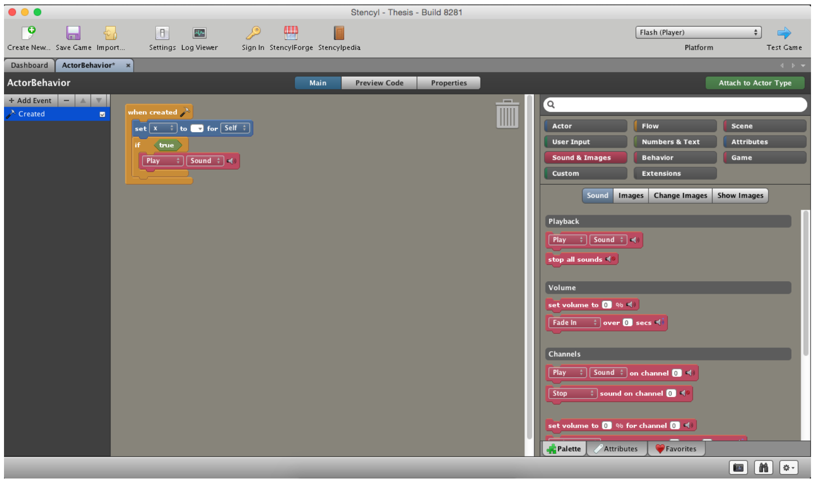


Figure 2-18: Stencyl Design Mode

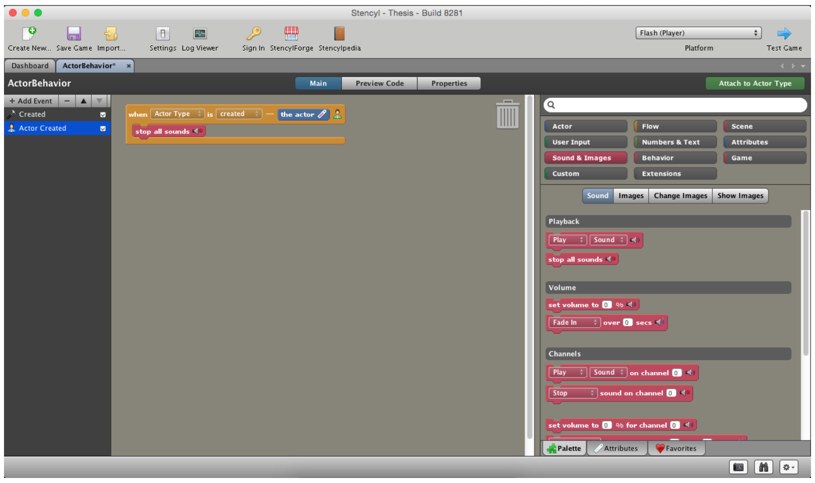


Figure 2-19: Each behavior in Stencyl has its own workspace

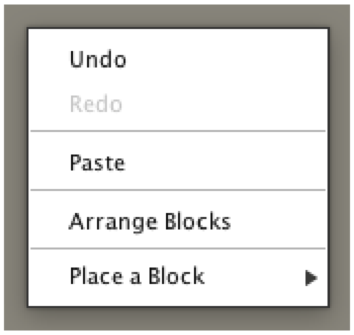


Figure 2-20: Users can arrange blocks using a right-click function

2.4.4 Gameblox

Gameblox "is a game editor that uses a block based programming language to allow anyone to make games" [6]. This platform, like App Inventor, uses the Blockly framework for its blocks editor.

A feature available in Gameblox, but not in Blockly or App Inventor, is blocks pages. Blocks pages are additional workspaces that users can create. These blocks pages are like the workspaces attached to sprites in Scratch and behaviors in Stencyl and allow users to transport blocks from one page to another. However, workspaces cannot be nested and explicit block organization within a singular workspace is not available.

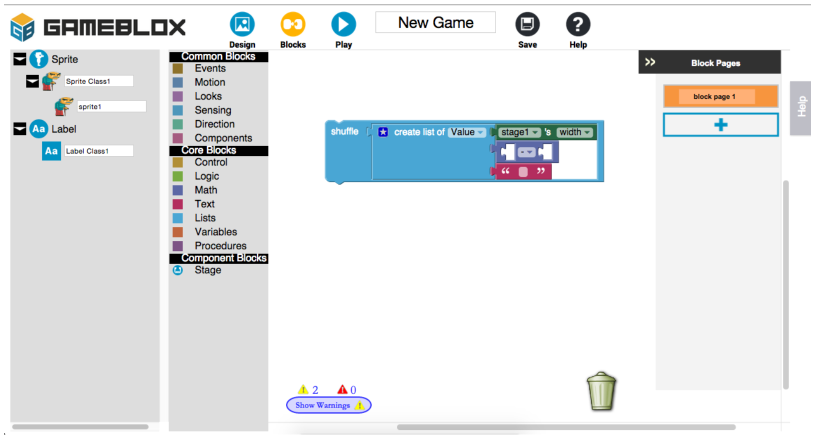


Figure 2-21: Gameblox blocks editor

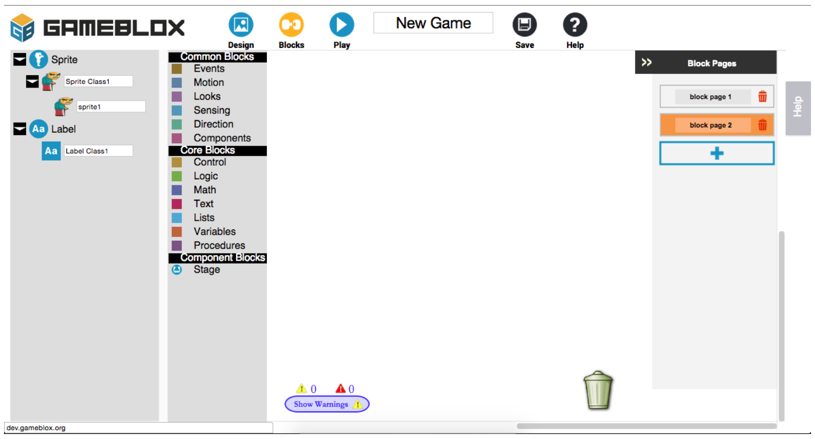


Figure 2-22: Additional block pages can be added

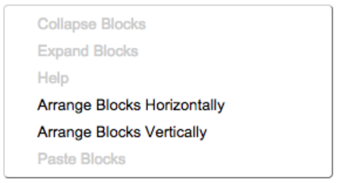
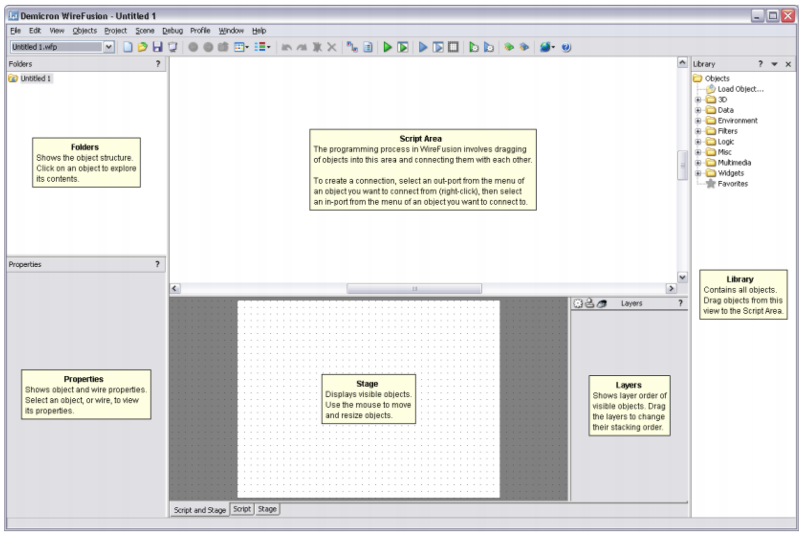


Figure 2-23: Users can arrange blocks using various right-click functions

2.4.5 WireFusion

WireFusion provides a tool for “quick creation of interactive 3D presentations for the internet” [4]. Interaction and functionality can be added to 3D models using WireFusion, which is not a blocks language but rather a dataflow language.

WireFusion has a singular 2D script area and does not provide users with the option of creating additional workspaces. Block are explicitly connected with each other with the means of arrows and their relationships are explicitly defined.



Folders
Shows the object structure. Click on an object to explore its contents.

Script Area
The programming process in WireFusion involves dragging of objects into this area and connecting them with each other.

To create a connection, select an out-port from the menu of an object you want to connect from (right-click), then select an in-port from the menu of an object you want to connect to.

Library
Contains all objects. Drag objects from this view to the Script Area.

Properties
Shows object and wire properties. Select an object, or wire, to view its properties.

Stage
Displays visible objects. Use the mouse to move and resize objects.

Layers
Shows layer order of visible objects. Drag the layers to change their stacking order.

Figure 2-24: Wirefusion work area

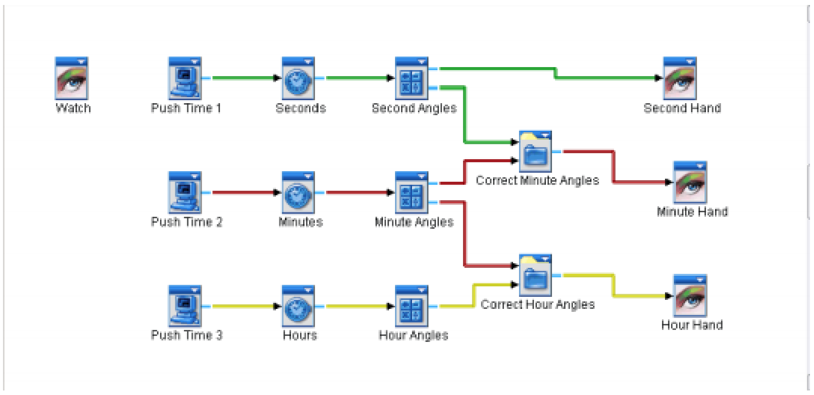


Figure 2-25: Wirefusion script area

Chapter 3

Design

One of the major ideas behind the design of Folder is habituation, or an action that becomes habit through constant use. Folders is seen through numerous operating systems and users are very familiar with its usage on a Desktop. Consistency between these applications is crucial [17, Chapter 1]. Gestures or actions that work on the Desktop but not in App Inventor will stress the user. In this chapter, the design choices of each action and functionality will be discussed in depth.

3.1 Overview

3.1.1 Adding a Folder

A Folder can be added to the workspace by dragging one out of the Folder drawer, similar to how a user can add a block to the workspace.

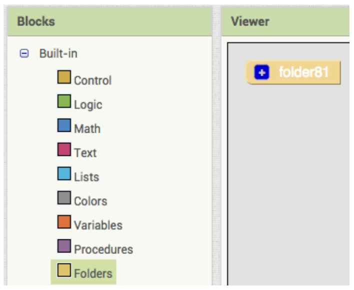


Figure 3-1: Adding a new Folder to the workspace

This is the biggest difference in design choice between Desktop folders and App Inventor Folders. On the Desktop, a folder is commonly created with a right-click action (Figure 3-2) but is also less commonly created using the File dropdown in the Menu bar (Figure 3-3). Dragging a Folder out from the Folder drawer is similar to creating a folder using the File Menu option but its design comes directly from how blocks are created in App Inventor. This was a deliberate choice made to better present the new feature to users. While a right-click menu option could have been viable, it would be not be as obvious a new feature to user. Since there was much demand for a tool like Folders, it seemed better to present it in a much more visible fashion, as a new Drawer.

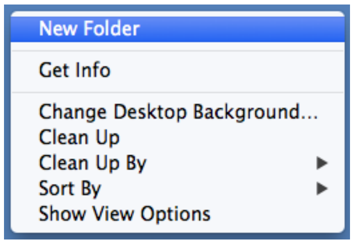


Figure 3-2: Creating a new folder on the desktop with a right-click

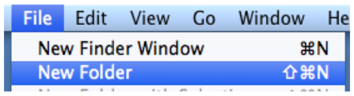


Figure 3-3: Creating a new folder on the desktop with the file menu

3.1.2 Expanding & Collapsing a Folder

Folders can exist in two states: collapsed and expanded. In its collapsed state, a Folder is a pseudo-block and presents itself like that of a block (Figure 3-4). A Folder is a pseudo-block in that it does not represent a code fragment and does not have the capability to form connections to other blocks. Displayed on the Folder pseudo-block is a [+] icon, indicating that the Folder is collapsed and can be expanded by pressing the icon. While clicking an icon to expand or collapse a Folder is not the expected behavior from a Desktop folder, it is consistent with blocks in App Inventor.

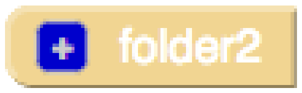


Figure 3-4: A collapsed Folder

In its expanded state, a Folder is both a pseudo-block and a mini-workspace (Figure 3-5).

When the [+] icon is pressed, the icon changes to [–] and the FOLDER's mini-workspace is displayed. If blocks have already been added to the FOLDER, these blocks will be displayed in the FOLDER's mini-workspace (Figure 3-6).

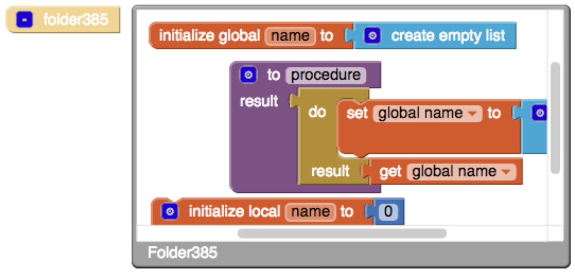


Figure 3-5: An expanded FOLDER

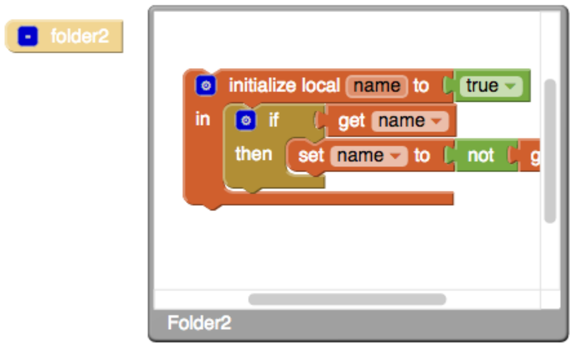


Figure 3-6: An expanded FOLDER with blocks in its mini-workspace

This design directly mimics that of a collapsed or expanded folder on any desktop computer (Figure 3-7 and 3-8).

Figure 3-7: A collapsed folder on a desktop

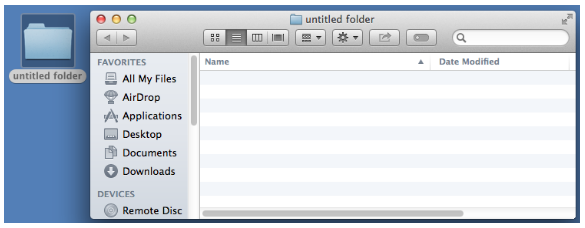


Figure 3-8: An expanded folder on a desktop

3.1.3 Deleting a Folder

A FOLDER can be deleted in the same ways that a block can be deleted: 1) right-click action (Figure 3-9), 2) drag over trash can (Figure 3-10, and 3) delete keyboard stroke. A block can be deleted in these same ways.

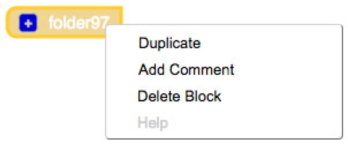


Figure 3-9: Right-click actions of a FOLDER

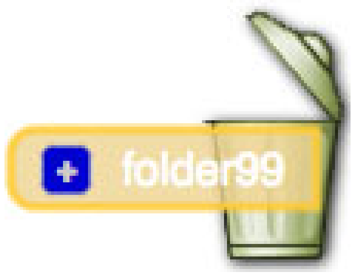


Figure 3-10: Deleting a FOLDER using the trash can

This design also mimics that of deleting a Desktop folder (Figures 3-11 and 3-12).

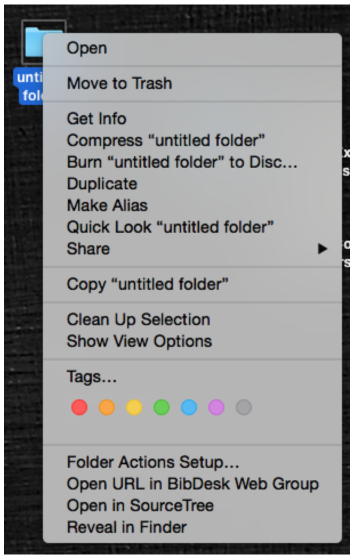


Figure 3-11: A folder can be deleted with a right-click drop down menu option

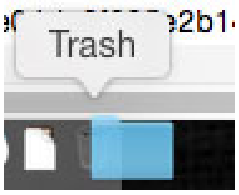


Figure 3-12: A folder can be deleted by dragging it to the trash

3.1.4 Adding a Comment to a Folder

A comment can be added to a Folder using a right-click action, similar to that of a block (Figure 3-9). Although adding comments to a folder is not a feature available on the Desktop, this is a feature available to blocks in App Inventor (Figure 3-13).

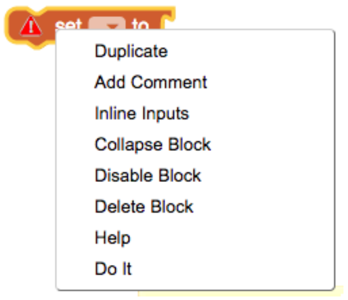


Figure 3-13: A comment can be added to a block in App Inventor

3.2 Editing Contents in a Folder

3.2.1 Moving a Block into a Folder

A block can be added into a Folder by clicking and dragging the block over the corresponding mini-workspace and letting go of the cursor (Figure 3-14). The mini-workspace the block will be placed in will be highlighted as visual feedback to the user. When mini-workspaces overlap in the workspace, the appropriate mini-workspace will be highlighted. In other words, if the block is added in the overlap region, the mini-workspace on top will be highlighted and the block will be added there.

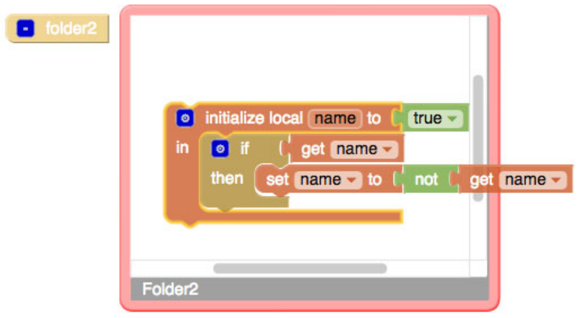


Figure 3-14: Moving a block into a Folder

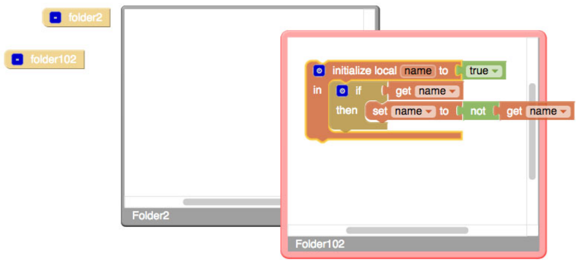


Figure 3-15: Highlighting of overlapping mini-workspaces

This behavior is also seen with folders on the Desktop (Figures 3-16 and 3-17).

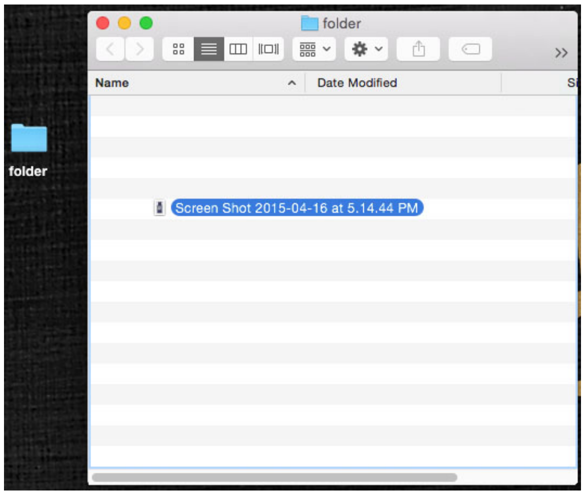


Figure 3-16: Moving a file into a folder

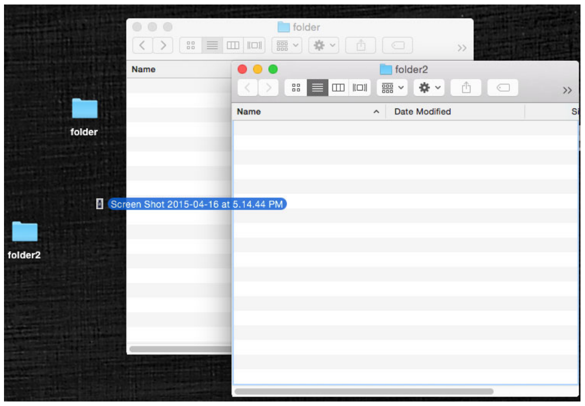


Figure 3-17: Highlighting of overlapping folders

3.2.2 Moving a Block out of a Folder

A block can be removed from a Folder in an identical process as moving a block into a Folder - click and drag the block over the new workspace, whether it is the main workspace or another mini workspace, and let go of the cursor (Figure 3-18). If no mini-workspaces are highlighted, the block will be placed in the main workspace. If a mini-workspace is highlighted, the block will be placed there.

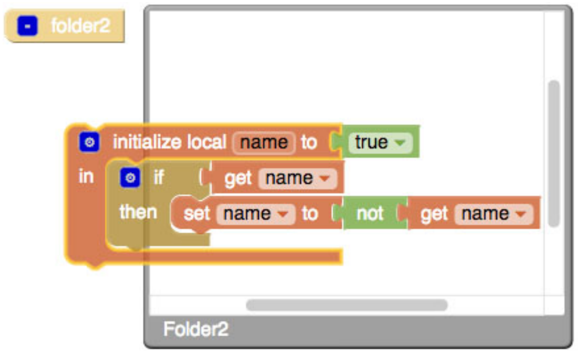


Figure 3-18: Moving a block out of a FOLDER

The behavior is similar but not identical on the Desktop. When a file is dragged from one folder to another, a semi-transparent copy of the file is moved (Figure 3-19. When the move is complete, the original file will move completely to the new space. Because of the strong colors and limited 2D available, I did choose to not imitate this behavior.

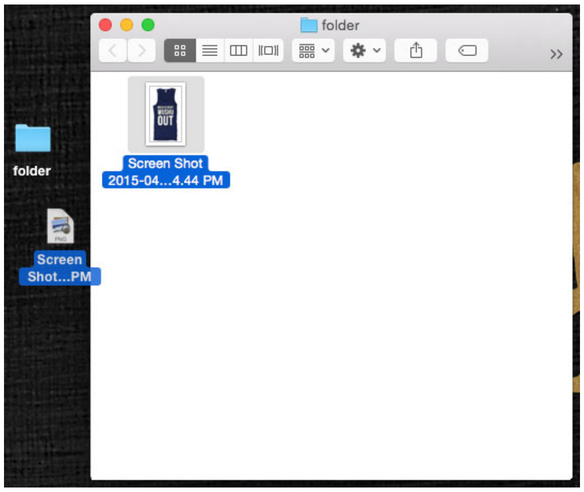


Figure 3-19: Moving a file out of a folder

3.2.3 Deleting Blocks in a Folder

Deletion of blocks inside or outside a FOLDER is the same process. A user can delete the block by 1) dragging it to the trash can, 2) using the delete keyboard stroke, or 3) right-click delete. This behavior is identical to deleting blocks from the main workspace, which is intended.

3.2.4 Connecting Blocks

The normal behavior of blocks when dragged is exhibited when blocks are dragged into or out of FOLDERS. In other words, when a block's connections are within the preset drag radius of a matching connection, the connections will highlight, indicating a viable connection. In a singular move, a block can be moved from one workspace to another and make a connection.

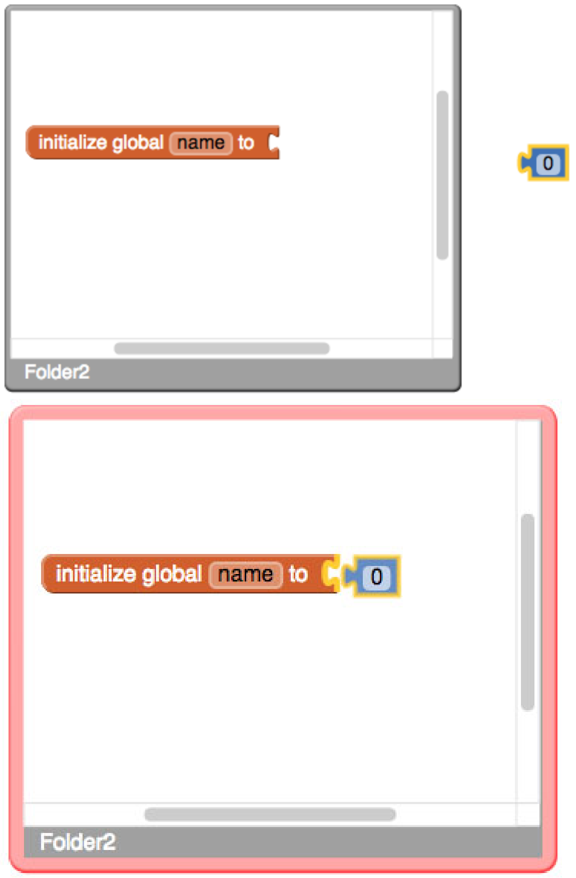


Figure 3-20: Moving a block and making a connection in one move

A similar, though different, functionality of Desktop folders is the ability to move a file from one workspace to another and add it to a nested folder in one move (Figure 3-21). This behavior shows that actions that work when two elements are in the same workspace should also work when the two elements are in two separate workspaces.

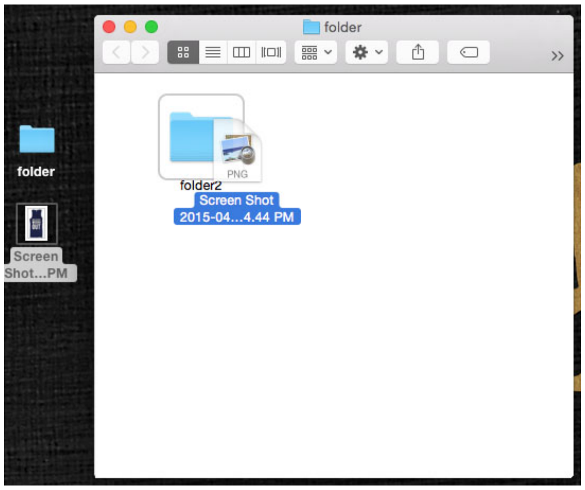


Figure 3-21: Moving a file to a nested folder in one move

3.3 Visual Aesthetics & Feedback

Visual feedback for the user is very important in blocks programming languages. If the expected behavior does not line up with the exhibited behavior, users can become frustrated. By giving users visual feedback, the expected behavior be the same as the exhibited behavior and if that is not the desired behavior, users have a hint of what to do to obtain the desired behavior.

3.3.1 Highlighting of the Mini-Workspace

One of the most important visual feedback elements of FOLDERS is highlighting of the mini-workspace on drag of blocks (Figures 3-14 and 3-15). Because of the visual feedback, users know exactly where their selected block will go, creating unambiguous behavior. The visual feedback is also seen on the Desktop (Figure 3-16).

3.3.2 Making Connections

Prior to FOLDERS, App Inventor highlights viable pairs of connections. This behavior is still exhibited with the addition of FOLDERS and adapted to connections made whilst adding or removing a block from a FOLDER (Figure 3-22). As mentioned previously, behavior exhibited when two elements are in the same workspace should also be exhibited when the elements are in two separate workspaces.

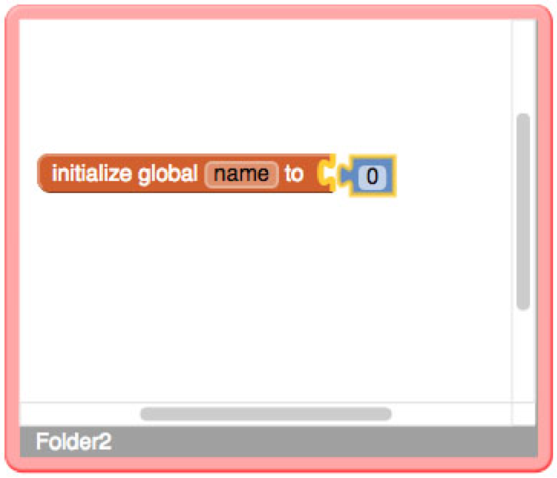


Figure 3-22: Highlighting of connections during a move

3.3.3 Visual Feedback vs Unexpected Behavior

In the example shown in Figure 3-23, some unexpected behavior is demonstrated. A block is dragged to the edge of a Folder, close to a matching connection. Without visual feedback, a user may expect the blocks to connect; however, the exhibited behavior would be the dragged block added to the Folder. Without highlighting of the mini-workspace, a user would be confused and frustrated. With the visual feedback and the realization that the connection is not highlighted while the mini-workspace is, the user understands why the exhibited behavior is exhibited and what to do to make the connection.

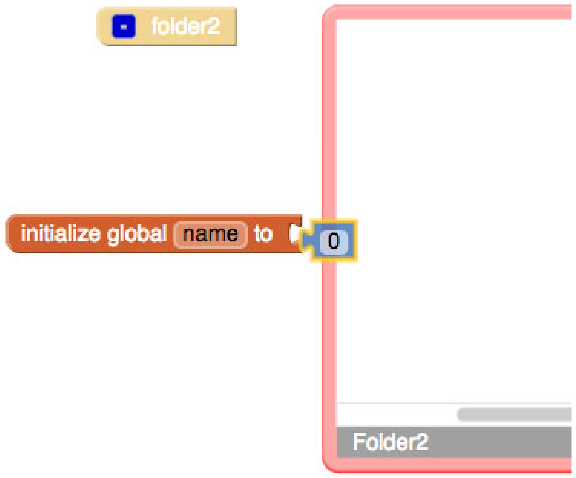


Figure 3-23: Visual feedback overcoming unexpected behavior

Using visual feedback to make ambiguous behavior clear is also seen with Desktop folders (Figure 3-24);

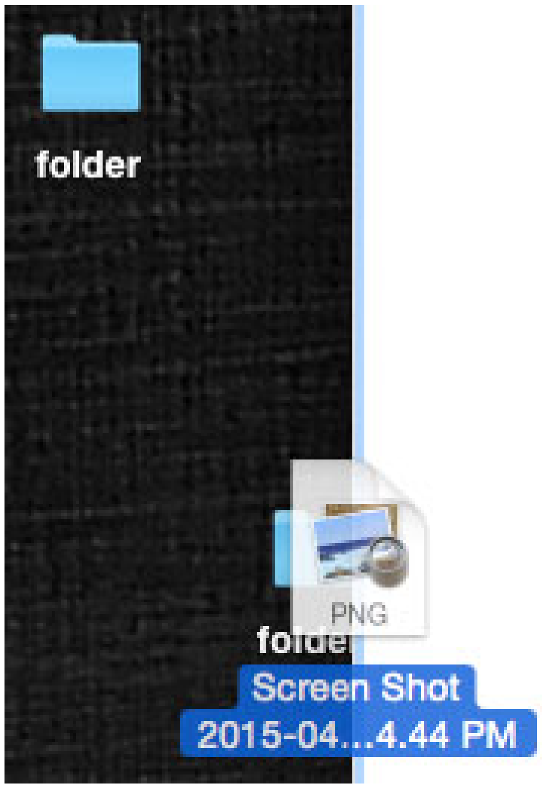


Figure 3-24: Visual feedback overcoming unexpected behavior on the Desktop

3.4 Semantics

Folders are purely a visual organizational tool for grouping blocks. While the visual semantics of the blocks in the workspace are manipulated, the semantics of the code fragments each block represents is unaffected. Blocks added to Folders are not abstracted and treated any different from blocks in the main workspace. Folders are not considered blocks and do not have code fragment counterparts. In other words, the final compiled product of App Inventor will not be any different when created with or without Folders.

Chapter 4

Implementation

MIT App Inventor's blocks editor uses the Blockly framework, which creates the blocks workspace using scalable vector graphics (SVG), as the foundation of the blocks workspace. Blockly handles the numerous nuisances of SVG using the closure library and Javascript classes. In this section, the various challenges of developing FOLDERS, as well as their solutions, will be discussed

The first challenge faced while developing for FOLDERS is the hesitation to change Blockly source code. Because App Inventor relies heavily on Blockly but does not directly contribute to Blockly's source code, changes to the Blockly source code do not carry through to Blockly updates. Therefore, any changes to the source code will have to be reimplemented with each large scale Blockly update. This challenge did not present itself as a true problem until the latter half of development.

4.1 Folder as a "Block"

4.1.1 Adding Folders to the Drawer Palette

The first step taken to create FOLDERS was defining a FOLDER "block". FOLDERS were first added to the Palette and then a FOLDER "block" was then added to the folders drawer. In order to this, 3 separate files were edited or added to appinventor/appengine/src/com/google/appinventor/:

1. images/folder.png
2. client/Images.java
3. client/boxes/BlockSelectorBox.java

folder.png is the beige icon representing the folder Palette item.

In Images.java, a new Palette item, folders, was added:

```
@Source("com/google/appinventor/images/folders.png")
ImageResource \folders();
```

In BlockSelectorBox.java, the Palette item FOLDERS was added to the array of drawer names and the beige square image of FOLDERS was linked with FOLDERS :

```
private static final String BUILTIN_DRAWER_NAMES[] = { "Control", "Logic", "Math", "
    Text",
"Lists", "Colors", "Variables", "Procedures", "\folders" };
```

```
bundledImages.put("\folders", images.\folders());
```

With these three changes, FOLDERS are now added to the list of built-in drawers in the Palette browser (Figure 4-1). Having added FOLDERS to the list of drawers, the FOLDER "block" needs to be created so instances of FOLDERS can be added to the workspace.

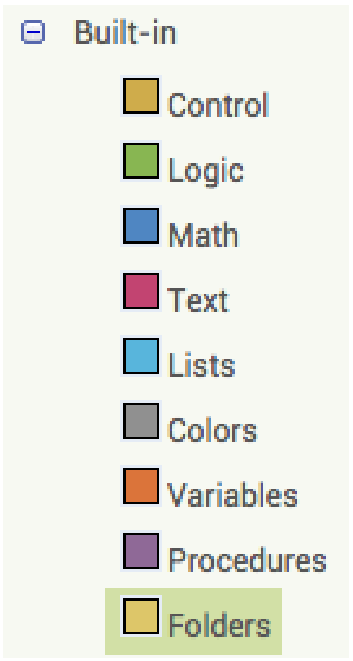


Figure 4-1: A new palette item - FOLDERS

4.1.2 Folder as a Block

FOLDERS was first a block before it became a pseudo-block. A FOLDER block had no connections and was labeled "FOLDER [the FOLDER 's id]" (e.g. folder1, folder20, etc). In order to add FOLDER as a block to the FOLDERS drawer, a new set of blocks was defined in the new file appinventor/blocklyeditor/src/blocks/folders.js:

```
'use strict';

goog.provide('Blockly.Blocks.folder');

Blockly.Blocks['folder'] = {
    category: "folders",
    init: function() {
        this.setColour(Blockly.FOLDER_CATEGORY_HUE);
        this.appendDummyInput()
            .appendField("\folder"+this.id);
    },
    typeblock: [{ translatedName: Blockly.Msg.LANG_FOLDERS_FOLDER }]
};
```

At this point, a FOLDER will appear in the folders drawer. This FOLDER instance will not yet have the [+]/[—] icons and is still considered a block. Up to this point, development for FOLDERS is fairly straightforward. FOLDERS has been considered a new set of blocks, with its own specialized drawer. The next step is to create FOLDERS as a pseudo-block and the path becomes complicated.

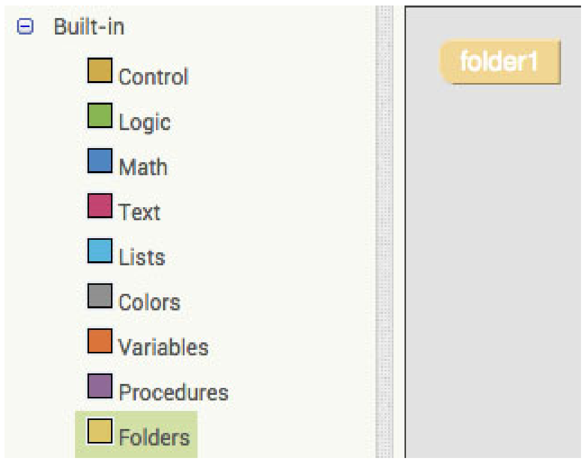


Figure 4-2: A new "block" - FOLDER

4.2 Folder , a Pseudo-Block

In order to determine the first step in developing FOLDERS as a pseudo-block, the differences between FOLDERS and blocks had to be considered. What makes FOLDERS special and what needs to be implemented to show this?

First, there needs to be a way to keep track of every FOLDERS in the workspace. Second, Folders each have a mini-workspace associated with it. Third, we need to keep track of whether the Folder is collapsed or expanded. With these three things in mind, development for Folders can begin.

Because Folders share most of blocks' functionality, folder.js inherits from block.js. In order to address the differences mentioned above, a few block functions needed to be rewritten for Folders: 1) obtain, 2) initialize, 3) fill, 4) getIcons, 5) initSvg, and 6) terminateDrag_. In order to keep track of all Folders in the workspace, a global variable Blockly.ALL_FOLDERS was created. Each Folder pseudo-block has a miniworkspace property, which points to the miniworkspace associated with the Folder. Another property added to Folders was the expandedFolder_ boolean which is toggled whenever the Folder icon is clicked. Each of the 6 methods mentioned above had, usually, small changes from their blocks counterpart in order to address these necessary Folder properties.

In addition to the defining the Folder class, a new folder_svg class must be defined. The folder_svg inherits from block_svg and has very few but important differences from block_svg. The most important difference is the creation of the folderIcon in the init method.

The implementation of folder.js can be found in Appendix B. Phase 1, which is described here, ends at line 171. The implementation of folder_svg.js can be found in Appendix C.

4.3 Folder Icon & Mini-Workspace

Development for the Folder icon and mini-workspace went through multiple iterations. The first consideration, which persisted through to the current iteration, is basing the Folder icon and mini-workspace off of the mutator icon and mutator bubble (Figure 4-3). This made sense because the mutator bubble is a mini-workspace and the mutator icon triggers the expansion and collapse of the mini-workspace.

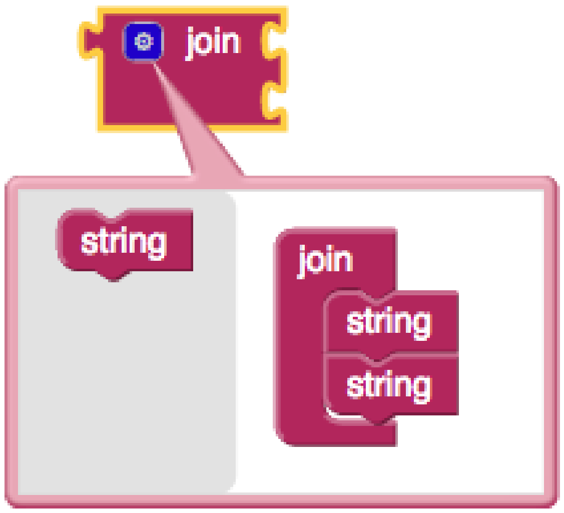


Figure 4-3: A text block with mutator icon and mutator bubble

4.3.1 Folder Icon

The process of creating the FOLDER icon class (folderIcon.js) was much simpler than creating the mini-workspace class. The FOLDER icon class is very similar to the mutator icon class (mutator.js), which inherits from the icon class (icon.js). The basic functionality of the FOLDER icon that needed to be created were:

1. Clicking the icon would collapse or expand the mini-workspace
2. Clicking the icon will toggle [+] to [–] and [–] to [+]
3. Clicking the icon will bring the FOLDER to the top of the ALL_FOLDERS list, which will allow the Blocks Editor to keep track of which FOLDER is visually on top
4. The above behavior is not executed when the FOLDER block is in a flyout

Behavior that needed to be reproduced from either the mutator icon or the general icon classes were:

1. Creating the SVG of icon
2. Displaying the icon in the correct location on the FOLDER block
3. Returning the location of the icon
4. Disposing the icon

5. Returning whether the icon is visible

Having these guidelines in mind, implementing the FOLDER icon class involved piecing together various methods from mutator.js and icon.js in order to satisfy the reproduction half, and creating or editing several methods to create the functionality specific for a FOLDER icon.

The implementation of folderIcon.js can be found in Appendix D.

4.3.2 Mini-Workspace

As mentioned before, development for the mini-workspace was much more complex. Originally, the mini-workspace was based off of the bubble class (bubble.js) but the bubble class had many methods which were unnecessary for a mini-workspace. Because a bubble is intended to pass mutation information to the parent block, these methods made sense for the bubble class but did not at all for a mini-workspace. Additionally, a bubble is anchored to its parent block but a mini-workspace should not be (Figure 4-4). Mini-workspaces could also be based off of flyouts, which are also some form of a mini-workspace. Again, flyouts came with too much functionality.

After much experimentation, it was determined that that the mini-workspace class should inherit from the workspace class because a mini-workspace is, at its essence, a workspace but with boundary constraints and a FOLDER hook.

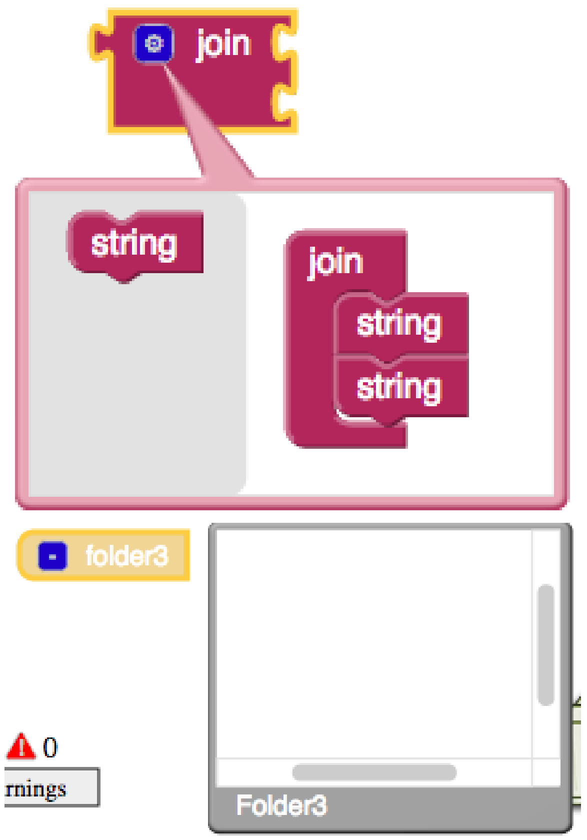


Figure 4-4: A bubble is anchored but a mini-workspace is not

The goal for this first iteration of the mini-workspace is to create the framework of a mini-workspace (Figure 4-5). The framework should look like a mini-workspace but not necessarily accept blocks or be scrollable.

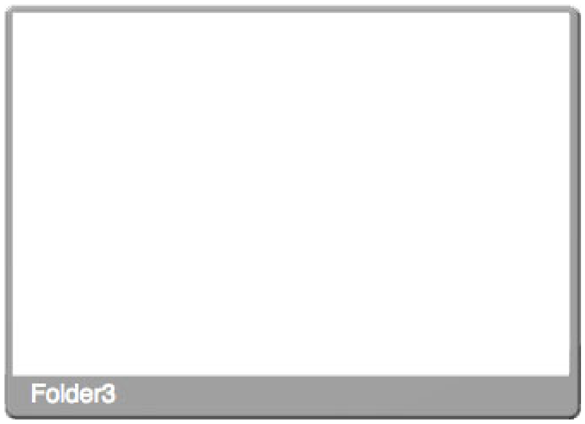


Figure 4-5: Mini-Workspace Framework

In order to this, the following functionality must be created:

1. A mini-workspace must remember its parent Folder, its top_blocks, whether it is a mini-workspace.
2. A mini-workspace needs to be rendered and disposed.
3. A mini-workspace needs to be rendered at both the Javascript and dom levels.
4. A mini-workspace needs to be able to be repositioned and not anchored to its Folder.
5. The mini-workspace must render and dispose as the folderIcon is clicked.

By implementing these functionalities, the mini-workspace framework was created. At this point, the mini-workspace does not accept blocks but can be moved around the main workspace and can expand (render) or collapse (dispose) as necessary.

4.4 Adding Functionality to Folders

4.4.1 Adding and Removing Blocks

At its most basic form, adding or removing blocks can be thought of as removing the block from the topBlocks_ of the old workspace and pushing it onto topBlocks_ of the new workspace. In order to implement this, the exact steps needed for a block to move from one workspace to another must be considered.

1. A block is clicked and an onMouseDown_ event is registered
2. The block is dragged and an onMouseDrag_ event is registered
3. The block is released on either a mini-workspace or a workspace and an onMouseUp_ event is registered

When the onMouseUp_ event is registered, the location of the cursor is used to determine what workspace the block is over. To do this, we traverse all the Folders in the workspace, which we saved in z-index order order in Blockly.ALL_FOLDERS. If the Folder is expanded, we can find the bounding box of the mini-workspace and determine whether the cursor is within this bounding box. Because the Folders are saved in z-index order, once the Folder is found, we need not continue looking.

```
Blockly.folder.prototype.isOverFolder = function(e) {
    if (this.expandedFolder_){
        var mouseXY = Blockly.mouseToSvg(e);
        var folderXY = Blockly.getSvgXY_(this.miniworkspace.svgGroup_);
        var width = this.miniworkspace.width_;
        var height = this.miniworkspace.height_;
        var over = (mouseXY.x > folderXY.x) &&
            (mouseXY.x < folderXY.x + width) &&
            (mouseXY.y > folderXY.y) &&
            (mouseXY.y < folderXY.y + height);
        return over;
    } else {
        return false;
    }
};
```

Listing 4.1: Blockly.Block.prototype.isOverFolder determines whether the cursor is over this FOLDER's mini-workspace

```
var overFolder = null;
      for (var i = 0; i < Blockly.ALL_FOLDERS.length; i++) {
        if (this_ != Blockly.ALL_FOLDERS[i] &&
            Blockly.ALL_FOLDERS[i].isOverFolder(e)) {
          overFolder = Blockly.ALL_FOLDERS[i];
          break;
        }
      }
```

Listing 4.2: A segment of code that finds the FOLDER that the mouse is over

Once the new workspace is found, we must remove the block from the old workspace's topBlocks_ array and push it onto the new workspace's topBlocks_ array. The last step needed is to rerender the both workspaces.

```
oldWorkspace.removeTopBlock(block);
newWorkspace.addTopBlock(block);
```

At this point, blocks can be moved from one workspace to another but the block would appear in the top right corner of the new workspace rather than where the cursor drops the block. This behavior will change with further development and details are laid out in Adding and Removing Blocks Part 2.

4.4.2 Saving Blocks

With the most basic and rudimentary behavior of FOLDERS completed, another important and crucial aspect of FOLDERS must be implemented - persistence of FOLDERS and its contents between sessions. Blocks in App Inventor projects are saved as XML in .bky files during auto-save and user triggered saves. A close look at xml.js was necessary.

A workspace is stored in the following XML format:

```
<xml>
  <block type="..." id="1" ...> ... </block>
  <block type="..." id="2" ...> ... </block>
</xml>
```

In order to store FOLDERS and their contents, I modified the XML to the following format with FOLDERS, where blocks with ids 4 and 5 are in the FOLDER's mini-workspace.

```
<xml>
  <block type="..." id="1" ...> ... </block>
  <block type="..." id="2" ...> ... </block>
  <block type="folder" id="3" >
    <block type="..." id="4" ...> ... </block>
    <block type="..." id="5" ...> ... </block>
  </block>
</xml>
```

To implement this, the code for workspaceToDom and domToWorkspace must both be changed. WorkspaceToDom was tackled first and the changes to the function are shown in lines 10-15 below:

```
1   Blockly.Xml.workspaceToDom = function(workspace) {
2     var width;  // Not used in LTR.
3     if (Blockly.RTL) {
4       width = workspace.getMetrics().viewWidth;
5     }
6     var xml = goog.dom.createDom('xml');
7     var blocks = workspace.getTopBlocks(true);
8     for (var i = 0, block; block = blocks[i]; i++) {
9       var element = Blockly.Xml.blockToDom_(block);
10        if (block.type == "folder") {
11            var folder = Blockly.Xml.workspaceToDom(block.miniworkspace);
```

```
     for (var x = 0, b; b = folder.childNodes[x];){
       element.appendChild(b);
     }
   }
  var xy = block.getRelativeToSurfaceXY();
  element.setAttribute('x', Blockly.RTL ? width - xy.x : xy.x);
  element.setAttribute('y', xy.y);
  xml.appendChild(element);
 }
 return xml;
};
```

Listing 4.3: Blockly.Xml.workspaceToDom

For domToWorkspace, a different approach was taken. FOLDERS are rendered collapsed and their mini-workspaces should not be rendered until expanded. Because of this, the xml of the mini-workspaces were saved as a variable of the mini-workspace rather than rendered immediately. This change can be seen in lines 19-25 below:

```
1  Blockly.Xml.domToWorkspace = function(workspace, xml) {
2   Blockly.Instrument.timer (
3     function () {
4      var width;  // Not used in LTR.
5      if (Blockly.RTL) {
6       width = workspace.getMetrics().viewWidth;
7      }
8  // The commented line below was replaced because it would reference beyond
9  // the end of the childNodes pseudo-array. In Chrome this is fine because
10 // the value returned is "undefined" which counts as false. However when
11 // using phantomjs (unit test) you wind up fetching memory garbage (!!)
12 //
13 //   for (var x = 0, xmlChild; xmlChild = xml.childNodes[x]; x++) {
14     var xmlChild;
15     for (var x = 0; x < xml.childNodes.length; x++) {
16      xmlChild = xml.childNodes[x];
17      if (xmlChild.nodeName.toLowerCase() == 'block') {
18       var block = Blockly.Xml.domToBlock(workspace, xmlChild);
19        if (block.type == "folder") {
20          var folderXML = goog.dom.createDom('xml');
21          while(xmlChild.children.length > 0) {
22            folderXML.appendChild(xmlChild.children[0]);
23          }
```

```
         block.miniworkspace.xml = folderXML;
       }
      var blockX = parseInt(xmlChild.getAttribute('x'), 10);
      var blockY = parseInt(xmlChild.getAttribute('y'), 10);
      if (!isNaN(blockX) && !isNaN(blockY)) {
       block.moveBy(Blockly.RTL ? width - blockX : blockX, blockY);
      }
     }
    }
   },
   function (result, timeDiff) {
    Blockly.Instrument.stats.domToWorkspaceCalls++;
    Blockly.Instrument.stats.domToWorkspaceTime = timeDiff;
   }
 );
};
```

Listing 4.4: Blockly.Xml.domToWorkspace

4.4.3 Compiling to Android

Along with saving blocks, compiling a project to an Android application is another essential aspect of Folders. App Inventor projects are first compiled to YAIL, which is then compiled into Java VM byte code. In order to compile correct Android applications, the proper YAIL must be generated.

In yail.js, the topBlocks_ of the main workspace is pulled and each of them generated into YAIL. TopBlocks_ of the main workspace do not, in fact, contain all of the topBlocks_ but rather contains the top blocks and Folders in the main workspace. In order to ensure that all of the topBlocks_ are in fact captured, we step through the topBlocks_ of the main workspace iteratively. If the current block is a Folder block, the topBlocks_ of its mini-workspace will be concatenated. The implementation can be seen in lines 13-21 below:

```
1  Blockly.Yail.getDebuggingYail = function() {
2   var code = [];
3   var componentMap = Blockly.Component.buildComponentMap([], [], false, false);
4
5   var globalBlocks = componentMap.globals;
6   for (var i = 0, block; block = globalBlocks[i]; i++) {
7    code.push(Blockly.Yail.blockToCode(block));
8   }
9
```

```
  var blocks = Blockly.mainWorkspace.getTopBlocks(true);
    //[Shirley 3/21] post-process of topBlocks

    var blocks2 = [];
    for (var x = 0, block; block = blocks[x]; x++) {
        if (block.category == "folders") {
            blocks2 = blocks2.concat(block.miniworkspace.topBlocks_);
        } else {
            blocks2 = blocks2.concat(block);
        }
    }
    blocks = blocks2;
    //[Shirley 3/21] end

  for (var x = 0, block; block = blocks[x]; x++) {

    // generate Yail for each top-level language block
    if (!block.category) {
      continue;
    }
    code.push(Blockly.Yail.blockToCode(block));
  }
  return code.join('\n\n');
};
```

Listing 4.5: Blockly.Yail.getDebuggingYail

4.4.4 Scrollable Mini-Workspace

Scrollable mini-workspaces was the next detail tackled. While non-scrollable mini-workspaces could have been an option, it did not make sense. First, the larger main workspace is scrollable. It would be strange to have one workspace scrollable and another not. Second, mini-workspaces are meant to give users more space to develop on. Non-scrollable mini-workspaces would limit the space available for the user and thus defeat the purpose of Folders.

Creating scrollable workspaces involved looking at how scrollbars were created for the main workspace and creating appropriate getWorkspaceMetrics_ and setWorkspaceMetrics_ functions. Because metrics is the gear driving the viewable window of any workspace, a lot of tweaking was necessary before scrollable workspaces were made possible.

The implementation for getWorkspaceMetrics_ and setWorkspaceMetrics_ can be seen below:

```
Blockly.MiniWorkspace.getWorkspaceMetrics_ = function () {
    var svgSize = Blockly.svgSize();
    //the workspace is just a percentage though.
    svgSize.width *= 0.4;
    svgSize.height *= 0.7;

    //We don't use Blockly.Toolbox in our version of Blockly instead we use drawer.js
    //svgSize.width -= Blockly.Toolbox.width;  // Zero if no Toolbox.
    svgSize.width -= 0;  // Zero if no Toolbox.
    var viewWidth = svgSize.width - Blockly.Scrollbar.scrollbarThickness;
    var viewHeight = svgSize.height - Blockly.Scrollbar.scrollbarThickness;
    try {
        var blockBox = this.getCanvas().getBBox();
    } catch (e) {
        // Firefox has trouble with hidden elements (Bug 528969).
        return null;
    }
    if (this.scrollbar_) {
        // Add a border around the content that is at least half a screenful wide.
        // Ensure border is wide enough that blocks can scroll over entire screen.
        var leftEdge = Math.min(blockBox.x - viewWidth / 2,
            blockBox.x + blockBox.width - viewWidth);
        var rightEdge = Math.max(blockBox.x + blockBox.width + viewWidth / 2,
            blockBox.x + viewWidth);
        var topEdge = Math.min(blockBox.y - viewHeight / 2,
            blockBox.y + blockBox.height - viewHeight);
        var bottomEdge = Math.max(blockBox.y + blockBox.height + viewHeight / 2,
            blockBox.y + viewHeight);
    } else {
        var leftEdge = blockBox.x;
        var rightEdge = leftEdge + blockBox.width;
        var topEdge = blockBox.y;
        var bottomEdge = topEdge + blockBox.height;
    }
    //We don't use Blockly.Toolbox in our version of Blockly instead we use drawer.js
    //var absoluteLeft = Blockly.RTL ? 0 : Blockly.Toolbox.width;
    var absoluteLeft = Blockly.RTL ? 0 : 0;
    var metrics = {
        viewHeight: svgSize.height,
        viewWidth: svgSize.width,
        contentHeight: bottomEdge - topEdge,
        contentWidth: rightEdge - leftEdge,
        viewTop: -this.scrollY,
        viewLeft: -this.scrollX,
        contentTop: topEdge,
        contentLeft: leftEdge,
        absoluteTop: 0,
        absoluteLeft: absoluteLeft
    };
    return metrics;
};

Blockly.MiniWorkspace.setWorkspaceMetrics_ = function(xyRatio) {
    if (!this.scrollbar) {
        throw 'Attempt to set mini workspace scroll without scrollbars.';
    }
    var metrics = this.getMetrics();//Blockly.MiniWorkspace.getWorkspaceMetrics_();
    if (goog.isNumber(xyRatio.x)) {
        this.scrollX = -metrics.contentWidth * xyRatio.x -
        metrics.contentLeft;
    }
    if (goog.isNumber(xyRatio.y)) {
        this.scrollY = -metrics.contentHeight * xyRatio.y -
        metrics.contentTop;
    }
    var translation = 'translate(' +
        (this.scrollX + metrics.absoluteLeft) + ',' +
        (this.scrollY + metrics.absoluteTop) + ')';
    this.getCanvas().setAttribute('transform', translation);
    this.getBubbleCanvas().setAttribute('transform',
        translation);
};
```

Listing 4.6: getWorkspaceMetrics_ and setWorkspaceMetrics_

4.4.5 Visual Feedback

As mentioned in the design of Folders, visual feedback is essential for the user in blocks programming languages. Without visual feedback and guidance, users can become frustrated when the exhibited behavior is not the expected behavior. In order to understand how to properly implement the visual highlighting of Folders, the highlighting of connections was studied closely.

The closest connection, if any, is found in the onMouseMove_ event handler for blocks. Using binary search, the closest connection is found and stored. If the current highlighted connection is not the same as the closest connection, the highlighted connection will be unhighlighted. If a closest connection is found and not currently highlighted, the connection will be highlighted. The code for this can been seen in Appendix G, Listing G.3, lines 72-100.

With this knowledge, the expanded FOLDER a block is over is found using the same methods, though with linear search and not binary search. While binary search is very important for connections because of the large number of connections that could be present on the workspace, it is unnecessary for FOLDERS as it is unlikely that a very large number of FOLDERS will be expanded on the workspace. The implementation for highlighting of mini-workspaces can be seen below:

```
Blockly.Block.prototype.onMouseMove_ = function(e) {
  ...
  //find the folder the block is over
  var overFolder = null;
  for (var i = 0; i < Blockly.ALL_FOLDERS.length; i++) {
    if (this_ != Blockly.ALL_FOLDERS[i] &&
      Blockly.ALL_FOLDERS[i].isOverFolder(e)) {
      overFolder = Blockly.ALL_FOLDERS[i];
      break;
    }
  }
  //remove highlighting if necessary
  if (Blockly.selectedFolder_ &&
    Blockly.selectedFolder_ != overFolder) {
    Blockly.selectedFolder_.miniworkspace.unhighlight_();
    Blockly.selectedFolder_ = null;
  }
  //add highlighting if necessary
  if (overFolder && overFolder != Blockly.selectedFolder_) {
    Blockly.selectedFolder_ = overFolder;
    Blockly.selectedFolder_.miniworkspace.highlight_();
  }
  ...
};
```

Listing 4.7: Blockly.Block.prototype.onMouseMove_

4.4.6 Adding and Removing Blocks Part 2

Having cleaned up the implementation of mini-workspaces and gained more knowledge about svg workspaces, a second iteration of adding and removing blocks was necessary. The following insights and solutions were made to implement the second iteration of moving a block from one workspace to another:

1. Blocks can only belong to one workspace. This first presented itself as a bug. A block inside a mini-workspace can be dragged around, but once the block is dragged over a border, it disappears. This is because the block is still in the mini-workspace and the mini-workspace extends past its visible border but is not viewable.

   Solution: A block's onMouseDown_ action will move the block to the main workspace. This will allow the block to be visible when dragged at all times.

2. Blocks exist on two separate but linked layers: dom and Javascript. A block's properties is stored as Javascript and this Javascript is used to generate the dom object. However, the dom can be manipulated easier and faster outside of Javascript.

   Insight: When a block is moved from one workspace to another, the dom of the block will be surgically moved in the dom. The Javascript properties of the block will be changed appropriately.

3. Blocks are not autonomous; they have parents and children. When blocks switch workspaces, their children must as well.

   Solution: By surgically moving the dom of a block, the children will be moved as well. On the Javascript side, the workspace of the block and its children will be changed recursively.

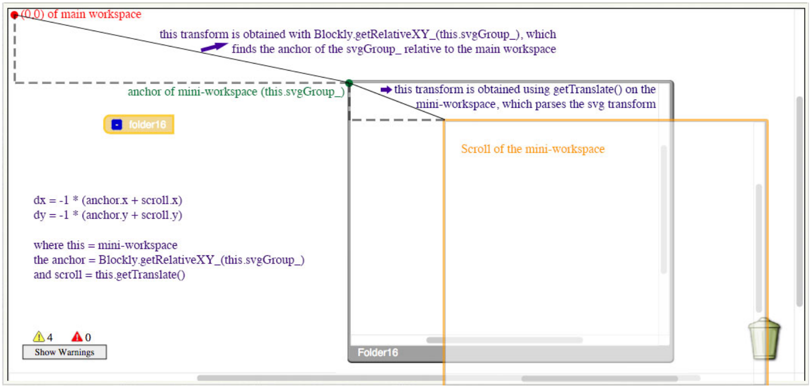
4. The block's new workspace is already stored. The implementation done with mini-workspace highlighting stored the highlighted mini-workspace in a global variable Blockly.selectedFolder_. This will considerably simplify the process of determining the new workspace.

   Insight: If the selected Folder is null, nothing needs to be done. The block is already in the main workspace. If the selected Folder is not null, the block needs to be moved to the selected Folder's mini-workspace.

5. The relationship of coordinates of a mini-workspace and the main workspace is related to the transform of the mini-workspace as well as the metrics, or scrolling transform, of the mini-workspace.

Insight: The relationship of the coordinates is shown in Figure 4-6.

Because this transform is needed for connections, as discussed in the next section, the implementation details will be revealed at the end of the connections discussion.


(0,0) of main workspace
anchor of mini-workspace (this.svgGroup_)
folder16
Scroll of the mini-workspace
4
0
Show Warnings
Folder16

this transform is obtained with Blockly.getRelativeXY_(this.svgGroup_), which finds the anchor of the svgGroup_ relative to the main workspace

this transform is obtained using getTranslate() on the mini-workspace, which parses the svg transform

```
dx = -1 * (anchor.x + scroll.x)
dy = -1 * (anchor.y + scroll.y)

where this = mini-workspace
the anchor = Blockly.getRelativeXY_(this.svgGroup_)
and scroll = this.getTranslate()
```

Figure 4-6: Relative coordinates of the main workspace and a mini-workspace

4.4.7 Connections

The last detail of FOLDERS tackled was connections. When a block is moved from mini-workspace to mini-workspace, its connections must also move. ConnectionDBLists are maintained for each workspace and their structure allows for easy binary search of viable connections, which directly leads to the possibility of highlighting connections.

There are 4 different types of connections: 1) value input, 2) value output, 3) statement previous, 4) statement next. The connectionDBList is a length 4 list, starting at index 1, with the each index referring the aforementioned type of connection.

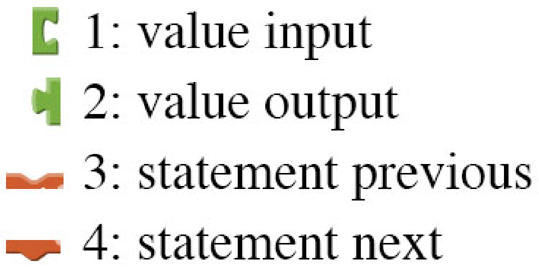


Figure 4-7: 4 different types of connections

At each of these indices, the list of connections are ordered first by y coordinate, then x coordinate.

A few careful steps must be taken to migrate connections from one workspace to another properly. First, every connection must be accounted for. Every connection on the top-most block and each of these connection's targetConnections must be migrated. This process must then be recursively repeated for every child block. Second, the x_ and y_ coordinate of each connection must be adjusted accordingly. The dx and dy are the same as the dx and dy of the previous section. This adjustment must also be done in a specific order: 1) the connection must first be removed from the old connectionDBList, 2) the x_ and y_ should be adjusted, 3) the connection is added to the new connectionDBList, and 4) the connection's dbList_ must be changed to the new workspace's connectionDBList.

The implementation for moving a block from the main workspace to the mini-workspace is as follows:

```
Blockly.Workspace.prototype.moveIntoFolder = function (block) {
  // The oldWorkspace will always be the mainWorkspace
  var oldWorkspace = Blockly.mainWorkspace;
  // newWorkspace will always be this
  var newWorkspace = this;

  // Move the Block into the right place in the \folder
  var blockRelativeToMWXY = block.getRelativeToSurfaceXY();
  var miniWorkspaceOrigin = Blockly.getRelativeXY_(this.svgGroup_);
  Blockly.mainWorkspace.removeTopBlock(block);
  this.addTopBlock(block);
  //surgically removes all svg associated with block from old workspace canvas
  var svgGroup = goog.dom.removeNode(block.svg_.svgGroup_);
  block.workspace = this;
  this.getCanvas().appendChild(svgGroup);

  var translate_ = this.getTranslate();
  var dx = -1 * (miniWorkspaceOrigin.x + parseInt(translate_[0]));
  var dy = -1 * (miniWorkspaceOrigin.y + parseInt(translate_[1]));
  var x = blockRelativeToMWXY.x + dx;
  var y = blockRelativeToMWXY.y + dy;
  block.svg_.getRootElement().setAttribute('transform',
      'translate(' + x + ', ' + y + ')');

  // remove, change x & y, add
  if (block.outputConnection) {
    changeConnection(block.outputConnection);
  }
  if (block.nextConnection) {
    changeConnection(block.nextConnection);
  }
  if (block.previousConnection) {
    changeConnection(block.previousConnection);
  }
  if (block.inputList) {
    for (var i = 0; i < block.inputList.length; i++) {
      var c = block.inputList[i];
      if (c.connection) {
        changeConnection(c.connection);
      }
    }
  }

  function changeConnection (connect) {
    oldWorkspace.connectionDBList[connect.type].removeConnection_(connect);
    connect.x_ += dx;
    connect.y_ += dy;
    newWorkspace.connectionDBList[connect.type].addConnection_(connect);
    if (connect.targetConnection) {
      var tconnect = connect.targetConnection;
      oldWorkspace.connectionDBList[tconnect.type].removeConnection_(tconnect);
      tconnect.x_ += dx;
      tconnect.y_ += dy;
      newWorkspace.connectionDBList[tconnect.type].addConnection_(tconnect);
      tconnect.dbList_ = newWorkspace.connectionDBList;
    }
    connect.dbList_ = newWorkspace.connectionDBList;
  }

};
```

Listing 4.8: Blockly.Workspace.prototype.moveIntoFolder

The implementation for moving a block from a mini-workspace to the main workspace is as follows:

```
Blockly.Workspace.prototype.moveOutOfFolder = function (block) {
  // this is used everytime a block is clicked - if it's in main, don't move it
  if (block.workspace == Blockly.mainWorkspace) {
    return;
  }

  //Move block into the right place in the main workspace
  var oldWorkspace = block.workspace;
  var newWorkspace = this;
  var blockRelativeToWXY = block.getRelativeToSurfaceXY();
  var miniWorkspaceOrigin = Blockly.getRelativeXY_(oldWorkspace.svgGroup_);
  oldWorkspace.removeTopBlock(block);
  newWorkspace.addTopBlock(block);
  //surgically removes all svg associated with block from old workspace canvas
  var svgGroup = goog.dom.removeNode(block.svg_.svgGroup_);
  block.workspace = newWorkspace;
  newWorkspace.getCanvas().appendChild(svgGroup);

  var translate_ = oldWorkspace.getTranslate();
  var dx = miniWorkspaceOrigin.x + parseInt(translate_[0]);
  var dy = miniWorkspaceOrigin.y + parseInt(translate_[1]);
  var x = blockRelativeToWXY.x + dx;
  var y = blockRelativeToWXY.y + dy;
  block.svg_.getRootElement().setAttribute('transform',
      'translate(' + x + ', ' + y + ')');
  block.isInFolder = false;

  // Change the old workspace and new workspace's connectionDBList
  if (block.outputConnection) {
    changeConnection(block.outputConnection);
  }
  if (block.nextConnection) {
    changeConnection(block.nextConnection);
  }
  if (block.previousConnection) {
    changeConnection(block.previousConnection);
  }
  if (block.inputList) {
    for (var i = 0; i < block.inputList.length; i++) {
      var c = block.inputList[i];
      if (c.connection) {
        changeConnection(c.connection);
      }
    }
  }

  function changeConnection (connect) {
    oldWorkspace.connectionDBList[connect.type].removeConnection_(connect);
    connect.x_ += dx;
    connect.y_ += dy;
    newWorkspace.connectionDBList[connect.type].addConnection_(connect);
    if (connect.targetConnection) {
      var tconnect = connect.targetConnection;
      oldWorkspace.connectionDBList[tconnect.type].removeConnection_(tconnect);
      tconnect.x_ += dx;
      tconnect.y_ += dy;
      newWorkspace.connectionDBList[tconnect.type].addConnection_(tconnect);
      tconnect.dbList_ = newWorkspace.connectionDBList;
    }
    connect.dbList_ = newWorkspace.connectionDBList;
  }

  newWorkspace.moveChild(block);

  return [dx,dy];

};
```

Listing 4.9: Blockly.Workspace.prototype.moveOutOfFolder

Chapter 5

Evaluation Methods

Folders were designed and implemented to improve the user experience and interface of App Inventor. In order to evaluate whether those goals were met, user studies must be conducted for Folders.

5.1 Initial Feedback

Upon the release of the first minimal viable product iteration of Folders, some informal testing and studies were completed. The Folder system was tested by App Inventor developers and the initial feedback was generally positive. During this alpha testing period, a number of bugs were discovered and a series of features were requested for future iterations of Folders.

Folders is considered to be an essential feature for App Inventor by all testers and a second iteration is greatly anticipated. Comparisons between Folders and blocks organization tools in other blocks languages, such as Gameblox, were made. There was some discussion on the merits of having an index displaying the tree structure of Folders in the blocks palette. The user interface of Folders also appeared to give several users trouble and is not as intuitive as expected.

5.2 Goals for User Study

With the feedback from the initial testing and studies, a set of goals can be set for a formal user study. The formal user study will be used to both gather data on the current iteration of Folders and establish the next steps for future iterations of Folders. Because Folders is an improvement on App Inventor, this study will be conducted on a voluntary sample of users already familiar with the layout, functionality, and features of App Inventor. Pending the results from this proposed study, a second study with new App Inventor users as subjects may be designed and carried out.

Phase one of the study will focus on the current iteration of Folders and its goals include determining whether the design is intuitive and whether the implementation exhibits the expected behavior. Phase two of the study aims at determining what additional features should be implemented for Folders and how essential do users consider these additional features.

5.3 Design of User Study

In order to achieves these goals, a mixture of survey and short-term longitudinal A/B testing will be conducted. The flow of the study will be as follows:

1. Users will be asked to complete a survey examining the expected behavior of Folders.
2. Users will be given a task to complete in live App Inventor. Development screens will be captured by screencast.
3. Users will be asked to evaluate the difficulty of the task in a second survey.
4. After two weeks, users will be asked to return and complete a similar task with one of many variations of Folders. Each variation will include or exclude a certain feature. Again, the development screens will be captured by screencast.
5. Users will be asked to evaluate the difficulty of the task and whether they found it more challenging or less challenging.

Questions I hope to answer with this user study include:

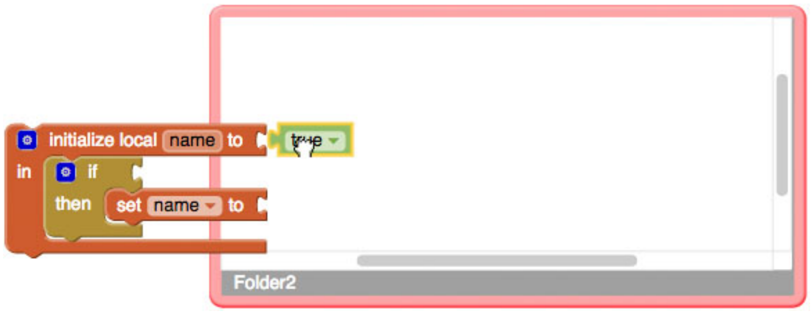
1. Are App Inventor programmers able to understand and use Folders without any guidance?
2. How necessary of a feature do App Inventor programmers consider Folders?
3. What behaviors, if any, of Folders is unexpected?
4. What features are still missing from Folders and how important are they?

5.3.1 Survey One

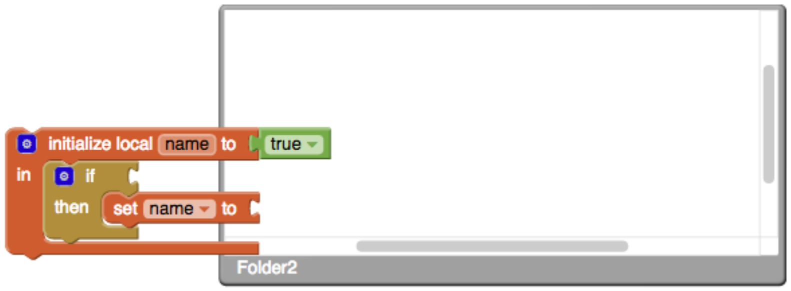
Question: Are the designs for Folders intuitive? Does Folders exhibit expected behavior?

This first survey will consistent of a series of scenario questions such as the following:

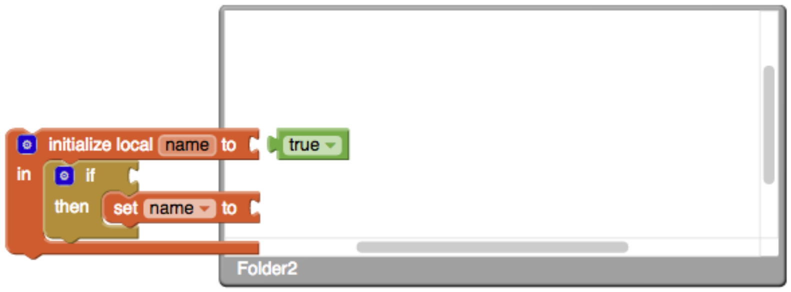
1. A user drags a boolean block within the expected connection radius of a block but also above an expanded FOLDER. What is the expected behavior?



a. The boolean block makes the connection.



b. The boolean block is put inside the FOLDER.



c. Other - please elaborate.

Each of these scenario questions will be targeted at answering a specific expected behavior question and aims at better understanding what users expect from a feature such as FOLDERS.

5.3.2 Control Behavior

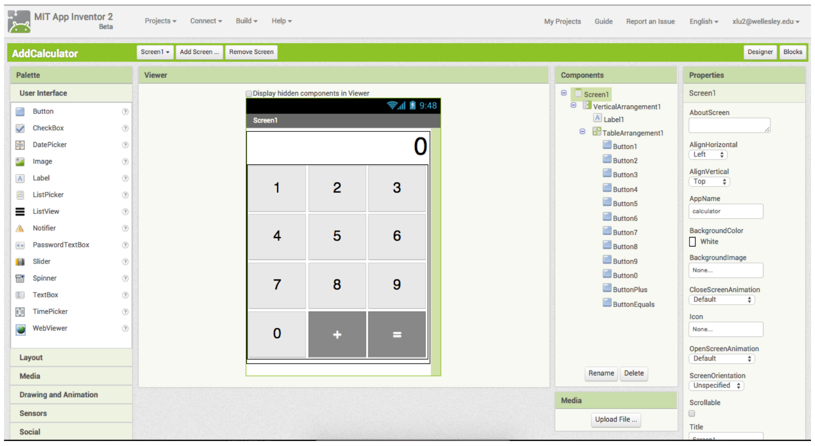
In order to measure whether FOLDERS is a useful tool, a control sample must be collected and compared to the treatment sample. Each user's development screen will be captured using a screen-cast tool. Each user will be given an App Inventor project file, with a completed design for the app they will build.

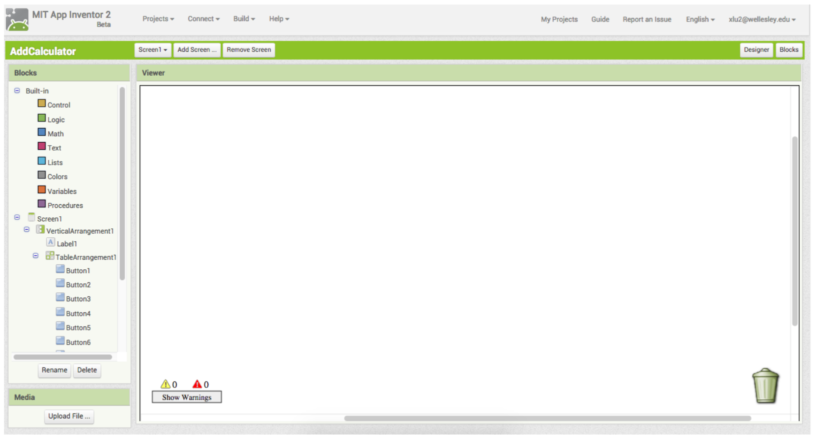
The task given to users will be:

Using the given AddCalculator.aia project file, create a completed Add Calculator. The Add Calculator should:

- Display the entered integers correctly
- Add two or more integers together accurately
- Behave as a normal calculator's addition functionality would

A physical calculator is available for behavior comparison purposes.





5.3.3 Exit Survey One

After completing the AddCalculator App, users will be given an open-ended exit survey consisting of the following questions:

1. How difficult did you find the AddCalculator app?
2. What would have aided you in development of the app?
3. What tools or user interface suggestions do you have for App Inventor?

5.3.4 Treatment Behavior

After two weeks, users will be asked to return and complete a second part of the study. In this study, users will be given a similar task but will be asked to use the current FOLDER version or a variation of the current FOLDER build of App Inventor. Variations can include:

- Blocks cannot be connected in the mini-workspace.
- Connection is prioritized over moving block into mini-workspace.
- No visual feedback through highlighting of the mini-workspace.
- A tree structure index of FOLDERS available to users in the Palette
- Naming available for FOLDERS
- Mini-workspace expands some distance from the FOLDER pseudo-block

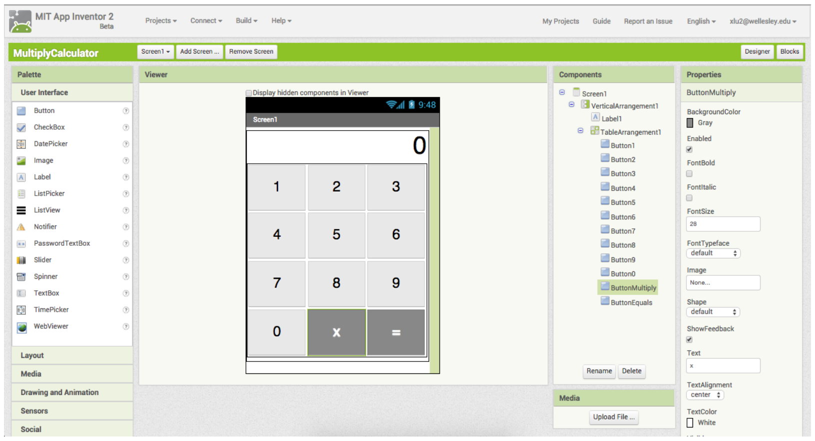
Before users are given the task, they will asked to rate on a scale of 1-5 how well they remember the task assigned to them two weeks prior. This will help assess the steps users took to complete this second task.

The task given to users will be:

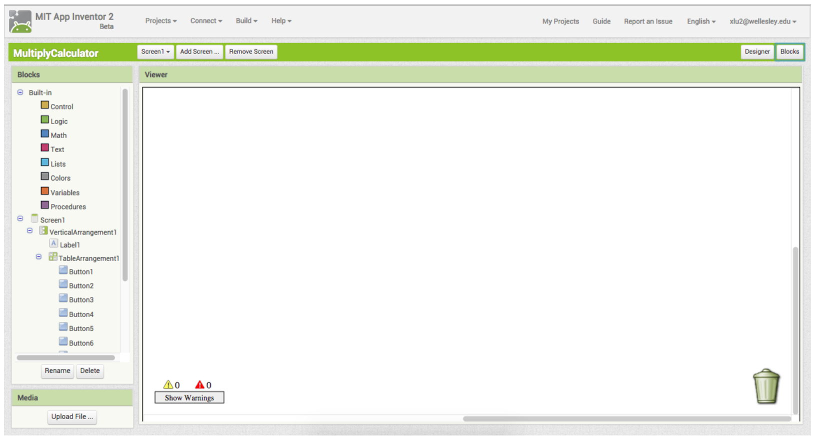
Using the given MultiplyCalculator.aia project file, create a completed Multiply Calculator. The Multiply Calculator should:

- Display the entered integers correctly
- Multiply two or more integers together accurately
- Behave as a normal calculator's multiplication functionality would

A physical calculator is available for behavior comparison purposes.



BackgroundColor: Gray
Enabled: ✓
FontBold: ☐
FontItalic: ☐
FontSize: 28
FontTypeface: default
Image: None...
Shape: default
ShowFeedback: ✓
Text: x
TextAlignment: center
TextColor: White



5.3.5 Exit Survey Two

Once users finish the Multiply Calculator app, a second open-ended exit survey will be given to users. Questions on the survey will include:

1. How difficult was developing Multiple Calculator?
2. Did FOLDERS help you during development? How?
3. Did you run into any bugs or unexpected behavior while using FOLDERS?
4. Would you recommend the usage of FOLDERS to other App Inventor developers? Why or why not?

5.3.6 Analysis of User Study

Upon completion of the user study, both the collected surveys and recorded screen-casts will be carefully reviewed and documented. The surveys will provide insight into how a user perceives FOLDERS and their thought process during development. The screen-casts will yield better understanding of the development process of a user and how they utilized, or did not utilize, FOLDERS in their tasks. By combining the results of the surveys and screen-casts, a clearer picture of the usefulness and demand of tool like FOLDERS will be presented.

Chapter 6

Conclusion and Future Work

6.1 Current State

In its current state, Folders work as a first iteration, minimal working product of a visual organization system for App Inventor. A Folder instance can be easily added to or deleted from the workspace. The mini-workspace associated with a Folder can expand and collapse its mini-workspaces. Blocks and other Folders can be added or removed from mini-workspaces. Blocks can be moved between workspaces and make connections in one action. There is visual feedback for the user as blocks are moved from one workspace to another to indicate which workspace the block is in. The design and behavior of Folders and mini-workspaces intentionally mimic the folders of a Desktop.

Some informal testing and studies have been conducted on this minimal viable product and their results will be driving future work and formal user studies for Folders.

6.2 Future Work

Future work will be discussed in 4 sections. First, the known bugs of Folders will be presented in a most severe to least severe fashion. These bugs were made known through the informal user testing and studies. Second, features for immediate development will be presented. These features are essential to Folders and must be implemented before Folders can be integrated into the production App Inventor system. Third, user studies and features for development the foreseeable future will be suggested. Each of these ideas require further brainstorming, development, and, ultimately, implementation. The implementation of any of these ideas will benefit Folders greatly.

Lastly, work for the distant future will be discussed. These ideas will be very important for Folders but also App Inventor.

6.2.1 Known Bugs

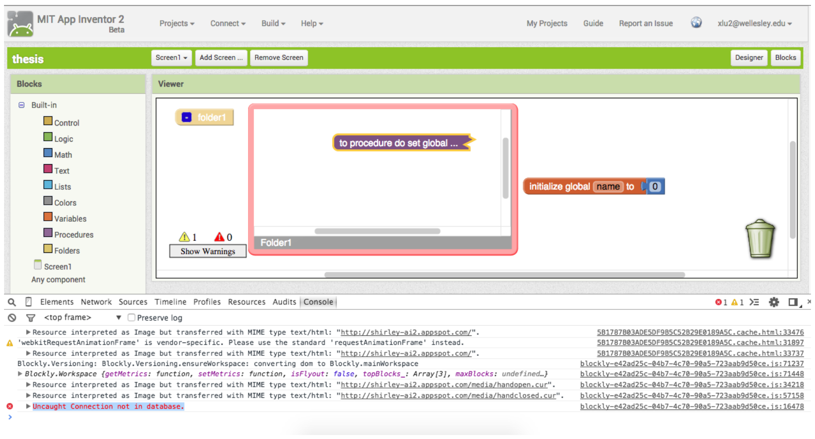
The current version of Folders has a number of bugs that need to be fixed before it can be introduced to the general App Inventor users. Below is a list of bugs, each of which has been given a rating from 1 to 4 with each rating corresponding to the following:

1. Critical Bug: breaks App Inventor; Folders cannot be pushed live until fixed
2. Major Bug: breaks Folders; does not break App Inventor; Folders cannot be integrated until fixed
3. Minor Bug: does not break App Inventor; exhibits unexpected behavior
4. Cosmetic Bug: does not break App Inventor; suboptimal user interface

6.2.1.1 Collapsed Blocks

Bug Rating: 1) Critical Bug

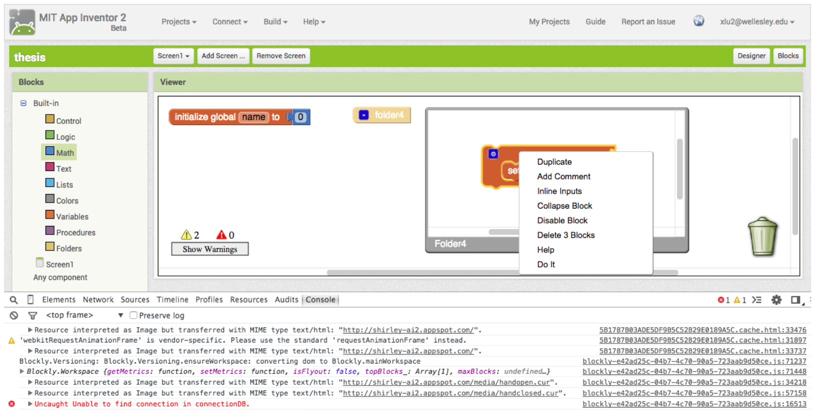
Collapsed blocks cannot be added to Folders at the moment. When a user tries to add a collapsed blocks to a Folder, a "Uncaught Connection not in database" is thrown. App Inventor will not exhibit any immediate problems but this will cascade into a number of other errors, ultimately leading to the loss of the original collapsed block.



```
▶Resource interpreted as Image but transferred with MIME type text/html: "http://shirley-ai2.appspot.com/".              5B1787B03ADE5DF9B5C52B29E0189A5C.cache.html:33476
'webkitRequestAnimationFrame' is vendor-specific. Please use the standard 'requestAnimationFrame' instead.              5B1787B03ADE5DF9B5C52B29E0189A5C.cache.html:31897
▶Resource interpreted as Image but transferred with MIME type text/html: "http://shirley-ai2.appspot.com/".              5B1787B03ADE5DF9B5C52B29E0189A5C.cache.html:33737
Blockly.Versioning: Blockly.Versioning.ensureWorkspace: converting dom to Blockly.mainWorkspace                      blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:71237
▶Blockly.Workspace {getMetrics: function, setMetrics: function, isFlyout: false, topBlocks_: Array[3], maxBlocks: undefined…}   blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:71448
▶Resource interpreted as Image but transferred with MIME type text/html: "http://shirley-ai2.appspot.com/media/handopen.cur".     blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:34218
▶Resource interpreted as Image but transferred with MIME type text/html: "http://shirley-ai2.appspot.com/media/handclosed.cur".   blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:57158
▶Uncaught Connection not in database.                                                                                blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:16478
>
```

Figure 6-1: Attempting to add a collapsed block will throw "Uncaught Connection not in database" error

Collapsing a block inside a mini-workspace will result in the same error being thrown; however, App Inventor will immediately display incorrect behavior. A forced refresh of App Inventor will be necessary and the original block will be lost.



```
▶Resource interpreted as Image but transferred with MIME type text/html: "http://shirley-ai2.appspot.com/".    5B1787B03ADE5DF9B5C52B29E0189A5C.cache.html:33476
'webkitRequestAnimationFrame' is vendor-specific. Please use the standard 'requestAnimationFrame' instead.    5B1787B03ADE5DF9B5C52B29E0189A5C.cache.html:31897
▶Resource interpreted as Image but transferred with MIME type text/html: "http://shirley-ai2.appspot.com/".    5B1787B03ADE5DF9B5C52B29E0189A5C.cache.html:33737
Blockly.Versioning: Blockly.Versioning.ensureWorkspace: converting dom to Blockly.mainWorkspace    blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:71237
▶Blockly.Workspace {getMetrics: function, setMetrics: function, isFlyout: false, topBlocks_: Array[1], maxBlocks: undefined…}    blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:71448
▶Resource interpreted as Image but transferred with MIME type text/html: "http://shirley-ai2.appspot.com/media/handopen.cur".    blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:34218
▶Resource interpreted as Image but transferred with MIME type text/html: "http://shirley-ai2.appspot.com/media/handclosed.cur".    blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:57158
▶Uncaught Unable to find connection in connectionDB.    blockly-e42ad25c-04b7-4c70-90a5-723aab9d50ce.js:16513
```

Figure 6-2: Attempting to collapse a block in a mini-workspace will throw “Uncaught Connection not in database” error

6.2.1.2 Nested Folders

Bug Rating: 2) Major Bug

If nested Folders are both expanded and the outer Folder is collapsed, the inner Folder cannot be collapsed.

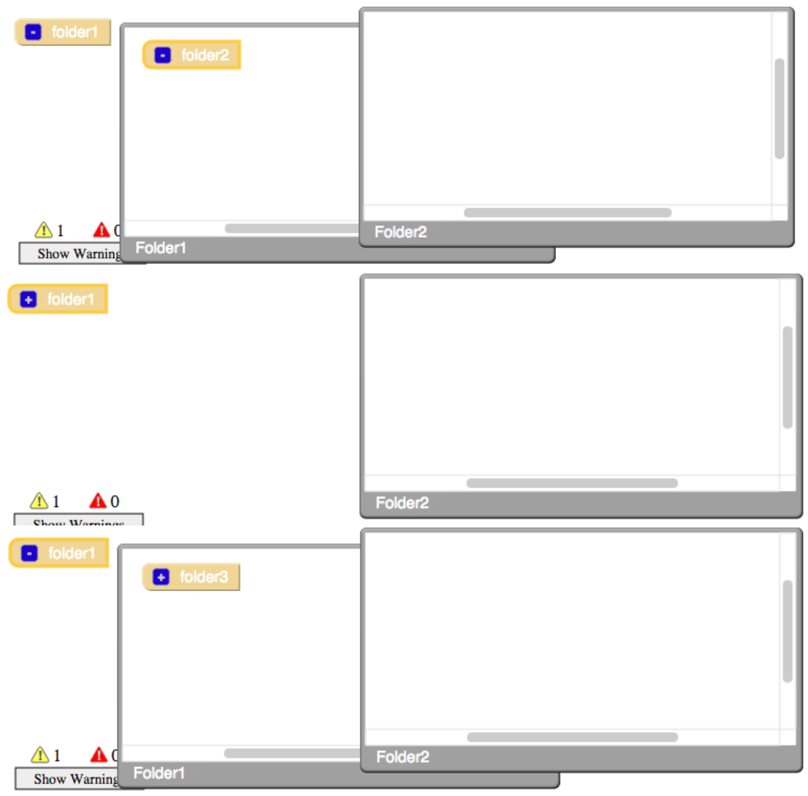


Figure 6-3: Inner nested mini-workspace cannot be collapsed

6.2.1.3 Variables & Procedures Not Recognized

Bug Rating: 2) Major Bug

Global variables and procedures inside FOLDERS are not recognized and will not be listed in Procedure and Variable drawers.

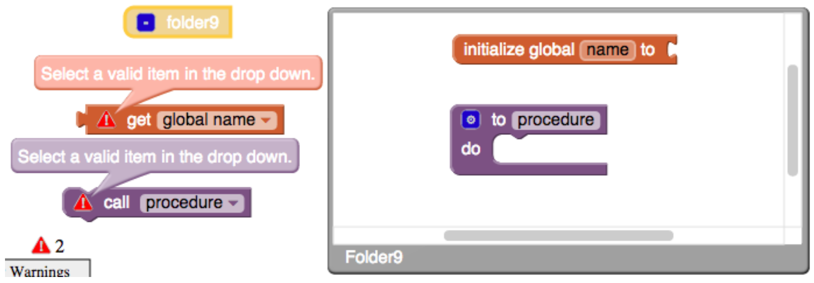

folder9
Select a valid item in the drop down.
get global name
Select a valid item in the drop down.
call procedure
2
Warnings
initialize global name to
to procedure
do
Folder9

Figure 6-4: Variables and procedures inside mini-workspaces are not recognized in the main workspace

Suggested Fix: edit workspace's getAllBlocks procedure. GetAllBlocks does not include blocks inside mini-workspaces. Alternatively, change procedure and variable's usage of getAllBlocks to include traversing all mini-workspaces as well.

6.2.1.4 Warnings Do Not Toggle

Bug Rating: 3) Minor Bug

Warnings and errors do not toggle inside a mini-workspace.

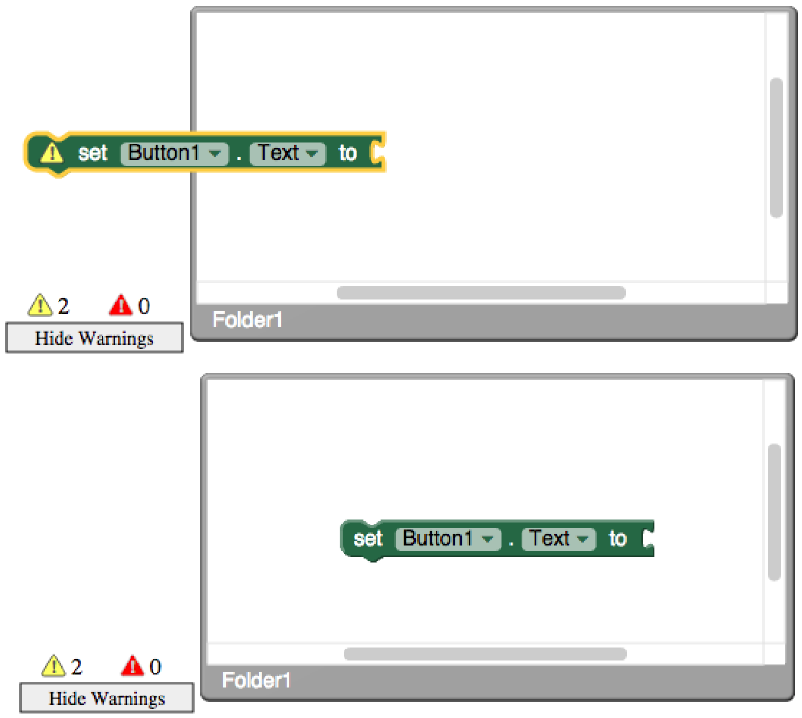


Figure 6-5: Warnings do not toggle for blocks inside mini-workspace

Suggested Fix: the block is no longer connected to the correct event; edit block events as part of workspace's moveIntoFolder and moveOutOfFolder methods.

6.2.1.5 Mutator Bubbles

Bug Rating: 3) Minor Bug

Mutator bubbles do not appear in the correct position when opened inside a mini-workspace.

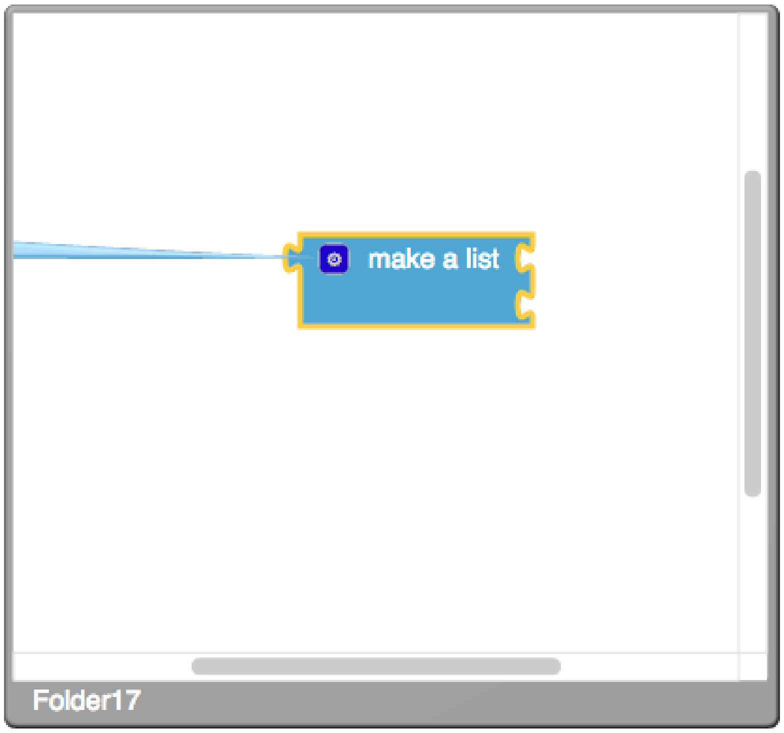


Figure 6-6: Opening a mutator bubble inside a mini-workspace results in unexpected behavior

6.2.1.6 Mutator Does Not Change Workspaces

Bug Rating: 3) Minor Bug

When a block with an expanded mutator is added to a mini-workspace, the mutator bubble remains in the main workspace.

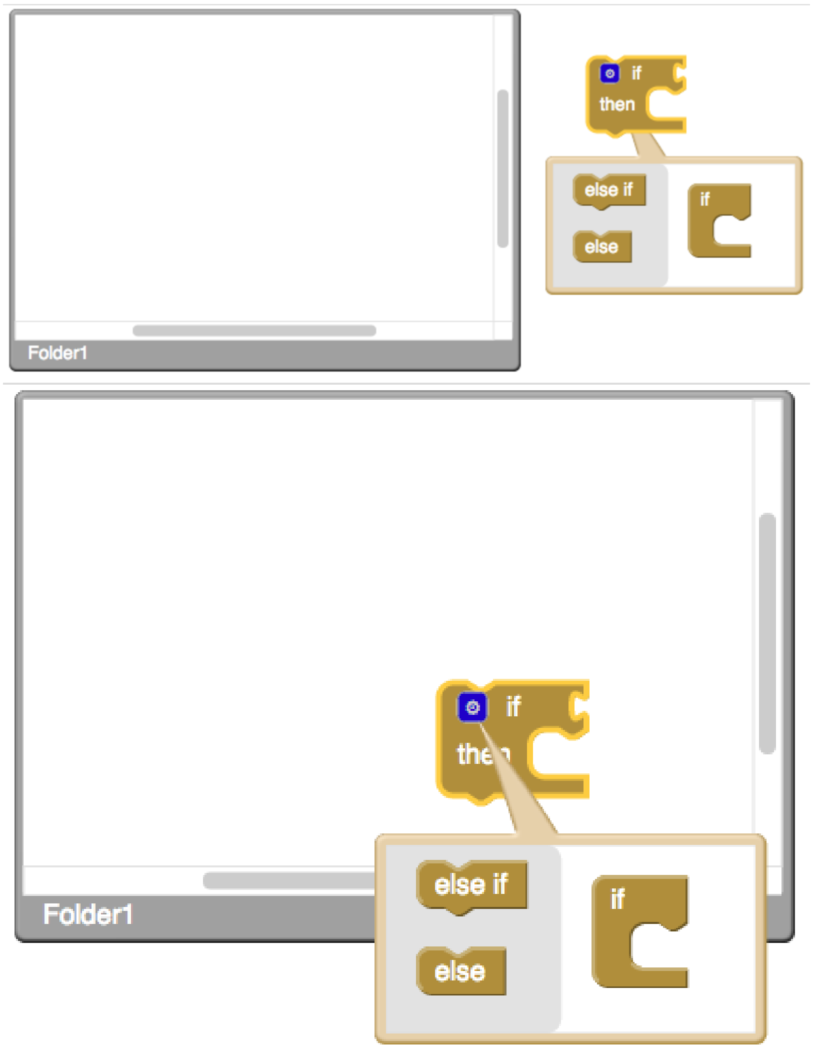


Figure 6-7: Mutator bubbles fail to migrate to the new workspace of the block

6.2.1.7 Folders Has Warning

Bug Rating: 3) Minor Bug

FOLDERS should not display "This block should be connected to an event block or a procedure definition" warning.

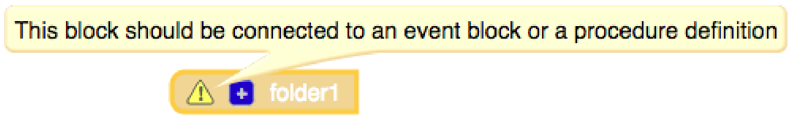

This block should be connected to an event block or a procedure definition

Figure 6-8: FOLDERS display warnings inappropriately

6.2.2 Immediate Future

There is a significant amount of work that needs to be done before Folders can be integrated into the production version of App Inventor. Some of this work has been mentioned in the Bugs List. A few features were requested in the early stage informal testing of Folders that require immediate attention and development.

6.2.2.1 Naming

Currently, Folders are named "Folder[block id]" (e.g. folder1, folder10, etc). The block id number changes with each rendering of a block, which is a feature inherent to Blockly. This misleads users into thinking that a new rendering of a specific Folder is a brand new Folder, which is not the case.

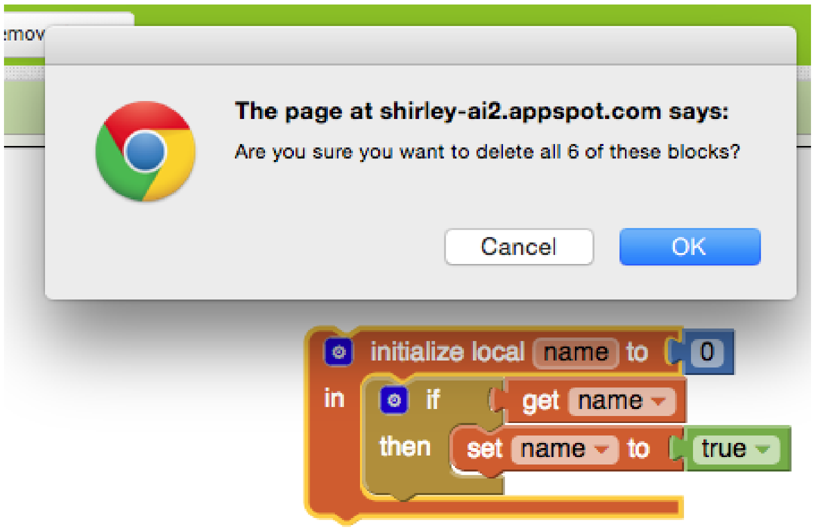
A feature that has been requested and expected by a number of different users is the ability to name Folders. By implementing a naming feature, users can name Folders with useful terms allowing for better navigation.

Implementation suggestion: create a new right-click drop down menu option for "Rename [Folder name]". Alternatively, change the Folder pseudo-block to include a text input.

6.2.2.2 Warning Before Deletion

Folders can be deleted without warning or confirmation. This is not ideal as Folders have the capacity to contain numerous crucial procedures and other blocks. Because App Inventor does not currently have an undo feature, deletion of a large number of blocks is not recoverable.

Implementation suggestion: a feature similar to that of deleting a large collection of blocks should be implemented for deleting a Folder with content.



The page at shirley-ai2.appspot.com says:

Are you sure you want to delete all 6 of these blocks?

Figure 6-9: Warning before deleting a block

6.2.2.3 Mini-workspace Expansion Anchor

Currently, a mini-workspace's expansion anchor is the upper-left corner of the FOLDER's icon. This is suboptimal as the [–] icon would covered and the way to collapse the mini-workspace is not immediately clear.

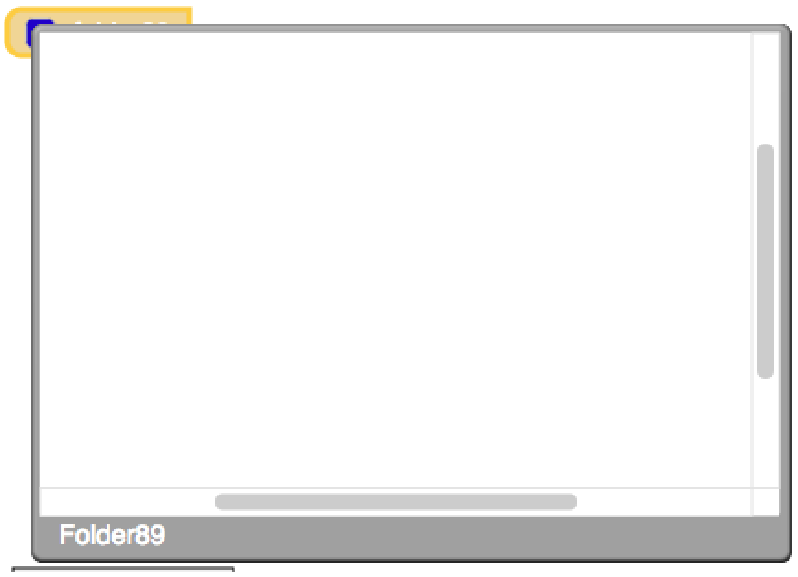


Figure 6-10: Mini-workspace expands on top of FOLDER pseudo-block

Implementation suggestion: the mini-workspace should expand some small distance away from the FOLDER pseudo-block such as in Figure 6-11.

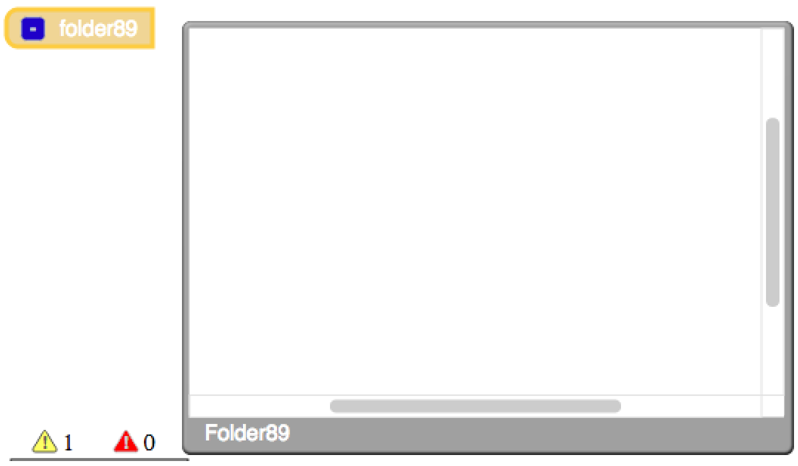


Figure 6-11: A better expansion of the mini-workspace

6.2.3 Foreseeable Future

6.2.3.1 User Studies

Informal testing has been conducted for the current minimal viable product version of FOLDERS. After critical bugs and features have been implemented for FOLDERS, I would like to see user studies conducted. A proposed user study has been described in Section 5.3 Design for User Study and its goals in Section 5.2 Goals for User Study.

6.2.3.2 Folders Duplication

Users may find the need to duplicate FOLDERS and its contents. Currently, duplicating a FOLDER will only create a new instance, without any of the original FOLDER's contents. Duplication of FOLDERS is an important feature because of its potential for users. For example, a FOLDER could contain the template for a series of blocks. By replicating the template FOLDER, users would not need to recreate these blocks one at a one.

6.2.3.3 Cumulative Error Display

Because FOLDERS hide away blocks for the user, it will be difficult for users to find where blocks with warnings or errors reside. By displaying a cumulative number of errors and warnings on blocks inside a FOLDER on the FOLDER pseudo-block, users will be given some direction in which to search for the offending block.

6.2.3.4 Recursive Disabling

For testing purposes, users may want to disable a series of blocks. If disabling a Folder indicates disabling all of the blocks nested inside the Folder, a user would not need to disable each block separately. This feature, although reasonable sounding, may not be the expected behavior and is pending user studies to determine its usability.

6.2.3.5 Resizable Mini-Workspace

One of the problems that led to the creation of Folders is the limited visible region of the main workspace. Constraining the viewable window of the mini-workspace defeats the purpose of giving the user more visible regions. Resizable mini-workspaces would also allow for easier development inside a Folder.

6.2.3.6 Collapse Button on Mini-Workspace

Because mini-workspaces are not anchored on its corresponding Folder pseudo-block, users may find it difficult identifying the correct Folder pseudo-block. If Folders are nested and collapsed, the problem is only exacerbated. By giving users the option to collapse the mini-workspace on the mini-workspace itself, searching for the mini-workspace's Folder would not become a problem. Additionally, this would better mimic the behavior of Folders on the Desktop.

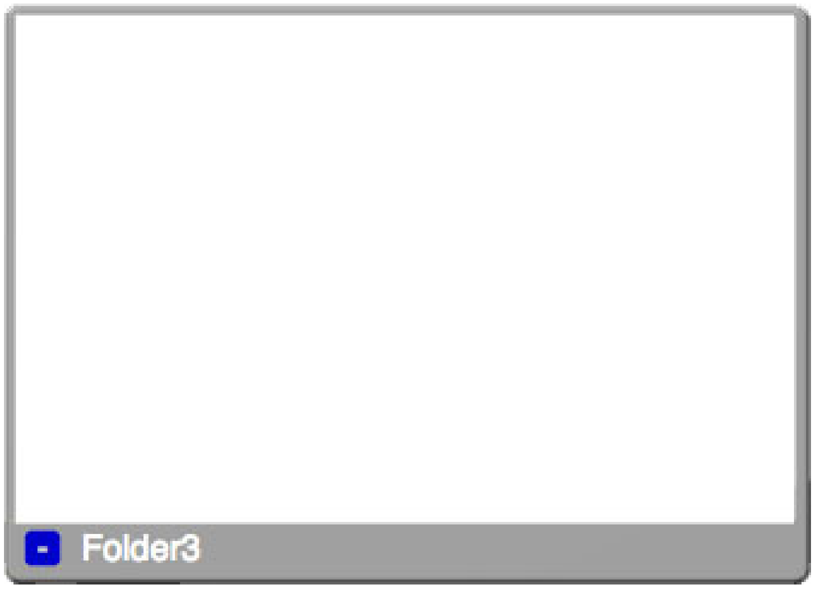


Figure 6-12: Design: collapse block button available on the border of mini-workspaces

6.2.3.7 Keyboard Controls

Using ctrl-c and ctrl-v to copy and paste is not available for blocks inside mini-workspaces. Because these options are available for blocks in the main workspace, it would be a disconnect in the user interface to have these options missing inside a mini-workspace. There are two options for where a pasted block should appear - inside the same FOLDER as the original block or in the main workspace. The better option may come as the result of a user study.

6.2.3.8 Folders Pseudo-Block Shortcut For Moving Blocks

On a desktop, files and other FOLDERS can be added to a FOLDER by dragging it over the FOLDER icon. This behavior is not replicated in FOLDERS for App Inventor but should be considered for future iterations of FOLDERS . One of the considerations for implementing this feature is the positioning of the new block or FOLDER and whether the new block or FOLDER should be allowed to overlap existing blocks or FOLDERS .

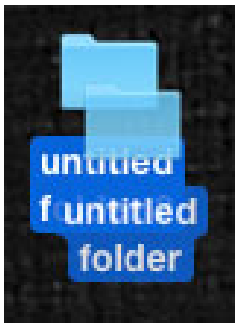


Figure 6-13: On the desktop, FOLDERS and files can be added by dragging it over the FOLDER icon

6.2.3.9 Folders Properties

FOLDERS are an excellent way for users to hide blocks away; however, it would take some effort for users to find specific blocks especially if there are a non-trivial number of FOLDERS in the workspace. FOLDERS properties could go a long way in this search. The properties could show what components are used in each FOLDER , how many blocks are in each FOLDER , a mini-map of the FOLDER 's mini-workspace, or a combination of these and more.

6.2.3.10 Right-Click Organization and Arrangement Tools For Mini-Workspace

Mini-workspaces should have the same workspace organization and arrangement tools as the main workspace (Figure 6-14). Any functionality available for the main workspace should be made available for a mini-workspace.

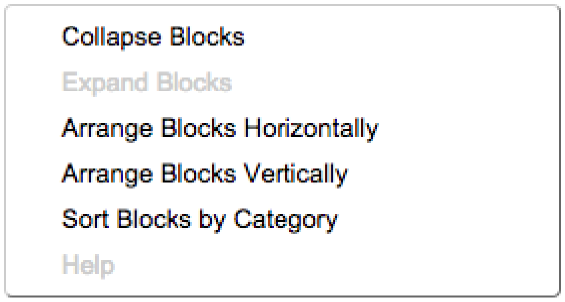


Figure 6-14: Workspace organization and arrangement tools for the main workspace

6.2.4 Distant Future

With the development of FOLDERS , a number of usages beyond the simple visual organization of blocks presents itself. Each of these usages can benefit blocks programming as a whole, especially with blocks programming as a teaching tool.

6.2.4.1 Abstraction Tool

FOLDERS as an abstraction tool is an expected direction for the organization tool to develop. FOLDERS, in some ways, already feel like an abstracted function. If the mechanism could accept parameters and in turn create different meanings for blocks nested inside the FOLDER, that would open up a whole new way of development in App Inventor. As a teaching tool, App Inventor can teach students abstraction explicitly rather than implicitly with the idea that each block represents a code fragment.

6.2.4.2 Sharing Mechanism

Similar to FOLDERS as an abstraction tool, FOLDERS as a sharing mechanism is another expected direction for the tool to take. FOLDERS, with additional development, can be used to share between screens, projects, and users.

6.2.4.3 Blocks Programming Coding Standards

With any textual language, there are a series of coding conventions, style guides, and best practices. For example, Javascript indentation is generally 2 spaces, variables are declared at the top of functions, and explicit scope should always be used. Unfortunately, blocks programming languages do not come with coding conventions, style guides, or best practices. This causes blocks code to vary greatly from project to project and user to user. This, in turn, causes navigating another user's code very difficult. With the introduction of FOLDERS and its clear, explicit, user-preference driven way of organizing blocks, coding standards can be established for App Inventor. Because App Inventor is a teaching tool, students can also be introduced to the idea of coding standards at an early stage.

Bibliography

[1] Alice.org. http://www.alice.org/. Accessed: 2015-04-22.

[2] MIT App Inventor 2. http://ai2.appinventor.mit.edu. Accessed: 2015-04-22.

[3] Scratch - Imagine, Program, Share. https://scratch.mit.edu/. Accessed: 2015-04-22.

[4] WireFusion - Realtime interactive 3D for internet marketing, 3D configurators and product visualizations. http://http://www.demicron.com/wirefusion/. Accessed: 2015-04-22.

[5] Stencyl - Make iPhone, iPad, Android & Flash games without code. http://http://www.stencyl.com/. Accessed: 2015-04-22.

[6] Gameblox. http://dev.gameblox.org/editor/. Accessed: 2015-04-22.

[7] Shieh K. Chi C. Huang, S. Factors affecting the design of computer icons. *International Journal of Industrial Ergonomics*, 2002.

[8] Phuwanartnurak A. J. Gill R. Bruce H. Jones, W. Don't take my folders away! organizing personal information to get things done. *Association for Computing Machinery*, 2005.

[9] Mitchell B. S. Rorres C. Mancoridis, S. Using automatic clustering to produce high-level system organizations of source code. *The International Wireless Industry Consortium*, 1998.

[10] R. S. Laramee. Bob's concise coding conventions. *Advances in Computer Science and Engineering*, 2013.

[11] A. Colangelo. The design of code: Organizing javascript. http://alistapart.com/article/the-design-of-code-organizing-javascript, 2013.

[12] P. Goodliffe. *Code Craft : The Practice of Writing Excellent Code.* No Starch Press, 2006.

[13] Parsons P. Dumas, J. Discovering the way programmers think about new programming environments. *Communications of the ACM*, 1995.

[14] Martin Fricke. *Logic and the Organization of Information.* Springer, 2012.

[15] Pencil Code. `https://pencilcode.net/`. Accessed: 2015-04-22.

[16] Scratch Wiki. `http://wiki.scratch.mit.edu/wiki/Scratch_Wiki_Home`. Accessed: 2015-04-22.

[17] J. Tidwell. *Designing Interfaces.* O'Reilly, 2011.

Appendix A

Folders.js

```
'use strict';

goog.provide('Blockly.Blocks.folder');

Blockly.Blocks['folder'] = {
    category: "Folders",
    init: function() {
        this.setColour(Blockly.FOLDER_CATEGORY_HUE);
        this.appendDummyInput()
            .appendField("folder"+this.id);
            //.appendField(new Blockly.FieldTextBlockInput('FOLDER NAME'), 'TEXT');
        //this.setMutator(new Blockly.Mutator(['procedures_mutatorarg']));
        this.setFolderIcon(new Blockly.FolderIcon());
    },
    decompose: function(workspace){
        return Blockly.decompose(workspace,'folder',this);
    },
    compose: Blockly.compose,
    typeblock: [{ translatedName: Blockly.Msg.LANG_FOLDERS_FOLDER }]
};
```

Listing A.1: appinventor/blocklyeditor/src/blocks/folders.js

Appendix B

Folder.js

```
'use strict';

goog.provide('Blockly.Folder');

goog.require('Blockly.Instrument'); // lyn's instrumentation code
goog.require('Blockly.FolderSvg');
goog.require('Blockly.Blocks');
goog.require('Blockly.Comment');
goog.require('Blockly.Connection');
goog.require('Blockly.ContextMenu');
goog.require('Blockly.ErrorIcon');
goog.require('Blockly.Input');
goog.require('Blockly.Msg');
goog.require('Blockly.Mutator');
goog.require('Blockly.Warning');
goog.require('Blockly.WarningHandler');
goog.require('Blockly.Workspace');
goog.require('Blockly.Xml');
goog.require('goog.Timer');
goog.require('goog.array');
goog.require('goog.asserts');
goog.require('goog.string');
goog.require('Blockly.Block');

Blockly.FOLDER_CATEGORY_HUE = [241, 213, 146];

Blockly.ALL_FOLDERS = [];

Blockly.Folder = function() {
    // We assert this here because there may be users of the previous form of
    // this constructor, which took arguments.
    goog.asserts.assert(arguments.length == 0,
        'Please use Blockly.Folder.obtain.');
};

goog.inherits(Blockly.Folder,Blockly.Block);

Blockly.Folder.obtain = function(workspace, prototypeName) {
    if (Blockly.Realtime.isEnabled()) {
        return Blockly.Realtime.obtainBlock(workspace, prototypeName);
    } else {
        var newFolder = new Blockly.Folder();
        newFolder.initialize(workspace, prototypeName);
        return newFolder;
    }
};

Blockly.Folder.prototype.initialize = function(workspace, prototypeName) {
    this.id = Blockly.genUid();
    workspace.addTopBlock(this);
    if (!workspace.isFlyout) {
        Blockly.ALL_FOLDERS.push(this);
    }
    this.fill(workspace, prototypeName);
    // Bind an onchange function, if it exists.
    if (goog.isFunction(this.onchange)) {
        Blockly.bindEvent_(workspace.getCanvas(), 'blocklyWorkspaceChange', this,
            this.onchange);
    }
};

Blockly.Folder.prototype.fill = function(workspace, prototypeName) {
    this.outputConnection = null;
    this.nextConnection = null;
    this.previousConnection = null;
    this.inputList = [];
    this.inputsInline = false;
    this.rendered = false;
    this.disabled = false;
    this.tooltip = '';
    this.contextMenu = true;

    this.parentBlock_ = null;
    this.childBlocks_ = [];
    this.deletable_ = true;
    this.movable_ = true;
    this.editable_ = true;
    this.collapsed_ = false;

    this.miniworkspace = new Blockly.MiniWorkspace(this,
        Blockly.MiniWorkspace.getWorkspaceMetrics_,
        Blockly.MiniWorkspace.setWorkspaceMetrics_);
    this.expandedFolder_ = false;
    this.workspace = workspace;
    this.isInFlyout = workspace.isFlyout;
    // This is missing from our latest version
    //workspace.addTopBlock(this);

    // Copy the type-specific functions and data from the prototype.
    if (prototypeName) {
        this.type = prototypeName;
        var prototype = Blockly.Blocks[prototypeName];
        goog.asserts.assertObject(prototype,
            'Error: "%s" is an unknown language block.', prototypeName);
        goog.mixin(this, prototype);
    }
    // Call an initialization function, if it exists.
    if (goog.isFunction(this.init)) {
        this.init();
    }
    // Bind an onchange function, if it exists.
    if ((!this.isInFlyout) && goog.isFunction(this.onchange)) {
        Blockly.bindEvent_(workspace.getCanvas(), 'blocklyWorkspaceChange', this,
            this.onchange);
    }
};

Blockly.Folder.prototype.getIcons = function() {
    var icons = [];
    if (this.mutator) {
        icons.push(this.mutator);
    }
    if (this.comment) {
        icons.push(this.comment);
    }
    if (this.warning) {
        icons.push(this.warning);
    }
    if (this.errorIcon) {
        icons.push(this.errorIcon);
    }
    if (this.folderIcon) {
        icons.push(this.folderIcon);
    }
    return icons;
};

Blockly.Folder.prototype.initSvg = function() {
    this.svg_ = new Blockly.FolderSvg(this);
    this.svg_.init();
    if (!Blockly.readOnly) {
        Blockly.bindEvent_(this.svg_.getRootElement(), 'mousedown', this,
            this.onMouseDown_);
    }
    this.workspace.getCanvas().appendChild(this.svg_.getRootElement());
};

Blockly.Folder.terminateDrag_ = function() {
    if (Blockly.Folder.onMouseUpWrapper_) {
        Blockly.unbindEvent_(Blockly.Folder.onMouseUpWrapper_);
        Blockly.Folder.onMouseUpWrapper_ = null;
    }
    if (Blockly.Folder.onMouseMoveWrapper_) {
        Blockly.unbindEvent_(Blockly.Folder.onMouseMoveWrapper_);
        Blockly.Folder.onMouseMoveWrapper_ = null;
    }
    var selected = Blockly.selected;
    if (Blockly.Folder.dragMode_ == 2) {
        console.log("terminate");
        // Terminate a drag operation.
        if (selected) {
            // Update the connection locations.
            var xy = selected.getRelativeToSurfaceXY();
            var dx = xy.x - selected.startDragX;
            var dy = xy.y - selected.startDragY;
            selected.moveConnections_(dx, dy);
            delete selected.draggedBubbles_;
            selected.setDragging_(false);
            selected.render();
            goog.Timer.callOnce(
                selected.bumpNeighbours_, Blockly.BUMP_DELAY, selected);
            // Fire an event to allow scrollbars to resize.
            Blockly.fireUiEvent(window, 'resize');
        }
    }
    if (selected) {
        selected.workspace.fireChangeEvent();
    }
    Blockly.Folder.dragMode_ = 0;
};

Blockly.Folder.prototype.removeFromAllFolders = function(folder) {
    var found = false;

    var index = this.indexOfFolder();
    if (index != -1){
        Blockly.ALL_FOLDERS.splice(index,1);
        found = true;
    }
    if (!found) {
        throw 'Folder not present in ALL_FOLDERS.';
    }
};

Blockly.Folder.prototype.indexOfFolder = function () {
    for (var f, x = 0; f = Blockly.ALL_FOLDERS[x]; x++) {
        if (f == this) {
            return x;
        }
    }
    return -1;
```

```
};

Blockly.Folder.prototype.setFolderIcon = function(folderIcon) {
    if (this.folderIcon && this.folderIcon !== folderIcon) {
        this.folderIcon.dispose();
    }
    if (folderIcon) {
        folderIcon.block_ = this;
        this.folderIcon = folderIcon;
        if (this.svg_) {
            folderIcon.createIcon();
        }
    }
};

Blockly.Folder.prototype.isOverFolder = function(e) {
    if (this.expandedFolder_){
        var mouseXY = Blockly.mouseToSvg(e);
        var folderXY = Blockly.getSvgXY_(this.miniworkspace.svgGroup_);
        var width = this.miniworkspace.width_;
        var height = this.miniworkspace.height_;
        var over = (mouseXY.x > folderXY.x) &&
            (mouseXY.x < folderXY.x + width) &&
            (mouseXY.y > folderXY.y) &&
            (mouseXY.y < folderXY.y + height);
        return over;
    } else {
        return false;
    }
};

Blockly.Folder.prototype.promote = function() {
    var index = this.indexOfFolder();
    var found = false;
    if (index != -1){
        found = true;
        Blockly.ALL_FOLDERS.splice(0, 0, Blockly.ALL_FOLDERS.splice(index, 1)[0]);
    }

    if (!found) {
        throw 'Folder not present in ALL_FOLDERS.';
        }
};
```

Listing B.1: appinventor/blocklyeditor/src/folder.js

Appendix C

Folder_svg.js

```
'use strict';

goog.provide('Blockly.FolderSvg');

goog.require('Blockly.Instrument'); // lyn's instrumentation code
goog.require('goog.userAgent');
goog.require('Blockly.BlockSvg');

Blockly.FolderSvg = function(folder) {
    this.block_ = folder;
    // Create core elements for the block.
    this.svgGroup_ = Blockly.createSvgElement('g', {}, null);
    this.svgPathDark_ = Blockly.createSvgElement('path',
        {'class': 'blocklyPathDark', 'transform': 'translate(1, 1)'},
        this.svgGroup_);
    this.svgPath_ = Blockly.createSvgElement('path', {'class': 'blocklyPath'},
        this.svgGroup_);
    this.svgPathLight_ = Blockly.createSvgElement('path',
        {'class': 'blocklyPathLight'}, this.svgGroup_);
    this.svgPath_.tooltip = this.block_;
    Blockly.Tooltip.bindMouseEvents(this.svgPath_);
    this.updateMovable();
};

goog.inherits(Blockly.FolderSvg,Blockly.BlockSvg);

Blockly.FolderSvg.prototype.init = function() {
    var folder = this.block_;
    this.updateColour();
    for (var x = 0, input; input = folder.inputList[x]; x++) {
        input.init();
    }
    if (folder.mutator) {
        folder.mutator.createIcon();
    }
    if (folder.folderIcon) {
        folder.folderIcon.createIcon();
    }
};
```

Listing C.1: appinventor/blocklyeditor/src/folder_svg.js

Appendix D

FolderIcon.js

```

'use strict';

goog.provide('Blockly.FolderIcon');
goog.require('Blockly.Folder');
goog.require('Blockly.MiniWorkspace');

Blockly.FolderIcon = function () {
    this.block_ = this;
    this.visible = false;
};

Blockly.FolderIcon.prototype.createIcon = function () {
    this.iconGroup_ = Blockly.createSvgElement('g', {}, null);
    this.block_.getSvgRoot().appendChild(this.iconGroup_);
    Blockly.bindEvent_(this.iconGroup_, 'mouseup', this, this.iconClick_);
    this.updateEditable();

    var quantum = Blockly.Icon.RADIUS / 2;
    var iconShield = Blockly.createSvgElement('rect',
        {'class': 'blocklyIconShield',
            'width': 4 * quantum,
            'height': 4 * quantum,
            'rx': quantum,
            'ry': quantum}, this.iconGroup_);
    this.iconMark_ = Blockly.createSvgElement('text',
        {'class': 'blocklyIconMark',
            'x': Blockly.Icon.RADIUS,
            'y': 2 * Blockly.Icon.RADIUS - 4}, this.iconGroup_);
    var icon = this.block_.expandedFolder_ ? "-" : "+";
    this.iconMark_.appendChild(document.createTextNode(icon));
};

Blockly.FolderIcon.prototype.renderIcon = function(cursorX) {
    if (this.block_.isCollapsed()) {
        this.iconGroup_.setAttribute('display', 'none');
        return cursorX;
    }
    this.iconGroup_.setAttribute('display', 'block');

    var TOP_MARGIN = 5;
    var diameter = 2 * Blockly.Icon.RADIUS;
    if (Blockly.RTL) {
        cursorX -= diameter;
    }
    this.iconGroup_.setAttribute('transform',
        'translate(' + cursorX + ', ' + TOP_MARGIN + ')');
    this.computeIconLocation();
    if (Blockly.RTL) {
        cursorX -= Blockly.BlockSvg.SEP_SPACE_X;
    } else {
        cursorX += diameter + Blockly.BlockSvg.SEP_SPACE_X;
    }
    return cursorX;
};

Blockly.FolderIcon.prototype.toggleIcon = function() {
    this.block_.expandedFolder_ = !this.block_.expandedFolder_;
    this.iconMark_.innerHTML = this.block_.expandedFolder_ ? "-" : "+";
};

Blockly.FolderIcon.prototype.iconClick_ = function(e) {
    this.toggleIcon();
    this.block_.promote();
    if (this.block_.isEditable()) {
        if (!this.block_.isInFlyout) {
            this.setVisible(!this.isVisible());
        }
    }
};

Blockly.FolderIcon.prototype.updateEditable = function() {

    if (this.block_.isEditable()) {
        // Default behaviour for an icon.
        if (!this.block_.isInFlyout) {
            Blockly.addClass_(/** @type {!Element} */ (this.iconGroup_),
                'blocklyIconGroup');
        } else {
            Blockly.removeClass_(/** @type {!Element} */ (this.iconGroup_),
                'blocklyIconGroup');
        }
    } else {
        // Close any mutator bubble.  Icon is not clickable.
        this.setVisible(false);
        Blockly.removeClass_(/** @type {!Element} */ (this.iconGroup_),
            'blocklyIconGroup');
    }
};

Blockly.FolderIcon.prototype.setVisible = function(visible) {
    if (visible == this.isVisible()) {
        // No change.
        return;
    }
    if (visible) {
        // Create the bubble.
        this.block_.miniworkspace.renderWorkspace(this.block_, this.iconX_, this.
            iconY_);
    } else {
        this.block_.miniworkspace.xml = Blockly.Xml.workspaceToDom(this.block_.
            miniworkspace);
        this.block_.miniworkspace.disposeWorkspace();
    }

    this.visible = !this.isVisible();
};

Blockly.FolderIcon.prototype.getIconLocation = function() {
    return {x: this.iconX_, y: this.iconY_};
};

Blockly.FolderIcon.prototype.dispose = function() {
    // Dispose of and unlink the icon.
    goog.dom.removeNode(this.iconGroup_);
    this.iconGroup_ = null;
    // Dispose of and unlink the bubble.
    this.setVisible(false);
    this.block_ = null;
};

Blockly.FolderIcon.prototype.computeIconLocation = function() {
    // Find coordinates for the centre of the icon and update the arrow.
    var blockXY = this.block_.getRelativeToSurfaceXY();
    var iconXY = Blockly.getRelativeXY_(this.iconGroup_);
    var  newX = blockXY.x + iconXY.x + Blockly.Icon.RADIUS;
    var newY = blockXY.y + iconXY.y + Blockly.Icon.RADIUS;
    if (newX !== this.iconX_ || newY !== this.iconY_) {
        this.setIconLocation(newX, newY);
    }
};

Blockly.FolderIcon.prototype.setIconLocation = function(x, y) {
    this.iconX_ = x;
    this.iconY_ = y;
};

Blockly.FolderIcon.prototype.isVisible = function () {
    return this.visible;
};
```

Listing D.1: appinventor/blocklyeditor/src/folderIcon.js

Appendix E

Miniworkspace.js

```
'use strict';

goog.provide('Blockly.MiniWorkspace');
goog.require('Blockly.Workspace');
goog.require('Blockly.ScrollbarPair');

Blockly.MiniWorkspace = function(folder,getMetrics,setMetrics) {
    Blockly.MiniWorkspace.superClass_.constructor.call(this, getMetrics, setMetrics);

    this.block_ = folder;
    this.topBlocks_ = [];
    this.maxBlocks = Infinity;
    this.svgGroup_ = null;
    this.svgBlockCanvas_ = null;
    this.svgBubbleCanvas_ = null;
    this.svgGroupBack_ = null;
    this.isMW = true;
};

goog.inherits(Blockly.MiniWorkspace, Blockly.Workspace);

Blockly.MiniWorkspace.prototype.rendered_ = false;
Blockly.MiniWorkspace.prototype.scrollbar_ = true;

Blockly.MiniWorkspace.prototype.anchorX_ = 0;
Blockly.MiniWorkspace.prototype.anchorY_ = 0;

Blockly.MiniWorkspace.prototype.relativeLeft_ = 0;
Blockly.MiniWorkspace.prototype.relativeTop_ = 0;
Blockly.MiniWorkspace.prototype.relativeLeft_ = 0;

Blockly.MiniWorkspace.prototype.width_ = 0;
Blockly.MiniWorkspace.prototype.height_ = 0;

Blockly.MiniWorkspace.prototype.autoLayout_ = true;

Blockly.MiniWorkspace.getWorkspaceMetrics_ = function () {
    var svgSize = Blockly.svgSize();
    //the workspace is just a percentage though.
    svgSize.width *= 0.4;
    svgSize.height *= 0.7;

    //We don't use Blockly.Toolbox in our version of Blockly instead we use drawer.js
    //svgSize.width -= Blockly.Toolbox.width;  // Zero if no Toolbox.
    svgSize.width -= 0;  // Zero if no Toolbox.
    var viewWidth = svgSize.width - Blockly.Scrollbar.scrollbarThickness;
    var viewHeight = svgSize.height - Blockly.Scrollbar.scrollbarThickness;
    try {
        var blockBox = this.getCanvas().getBBox();
    } catch (e) {
        // Firefox has trouble with hidden elements (Bug 528969).
        return null;
    }
    if (this.scrollbar_) {
        // Add a border around the content that is at least half a screenful wide.
        // Ensure border is wide enough that blocks can scroll over entire screen.
        var leftEdge = Math.min(blockBox.x - viewWidth / 2,
            blockBox.x + blockBox.width - viewWidth);
        var rightEdge = Math.max(blockBox.x + blockBox.width + viewWidth / 2,
            blockBox.x + viewWidth);
        var topEdge = Math.min(blockBox.y - viewHeight / 2,
            blockBox.y + blockBox.height - viewHeight);
        var bottomEdge = Math.max(blockBox.y + blockBox.height + viewHeight / 2,
            blockBox.y + viewHeight);
    } else {
        var leftEdge = blockBox.x;
        var rightEdge = leftEdge + blockBox.width;
        var topEdge = blockBox.y;
        var bottomEdge = topEdge + blockBox.height;
    }
    //We don't use Blockly.Toolbox in our version of Blockly instead we use drawer.js
    //var absoluteLeft = Blockly.RTL ? 0 : Blockly.Toolbox.width;
    var absoluteLeft = Blockly.RTL ? 0 : 0;
    var metrics = {
        viewHeight: svgSize.height,
        viewWidth: svgSize.width,
        contentHeight: bottomEdge - topEdge,
        contentWidth: rightEdge - leftEdge,
        viewTop: -this.scrollY,
        viewLeft: -this.scrollX,
        contentTop: topEdge,
        contentLeft: leftEdge,
        absoluteTop: 0,
        absoluteLeft: absoluteLeft
    };
    return metrics;
};

Blockly.MiniWorkspace.setWorkspaceMetrics_ = function(xyRatio) {
    if (!this.scrollbar) {
        throw 'Attempt to set mini workspace scroll without scrollbars.';
    }
    var metrics = this.getMetrics();//Blockly.MiniWorkspace.getWorkspaceMetrics_();
    if (goog.isNumber(xyRatio.x)) {
        this.scrollX = -metrics.contentWidth * xyRatio.x -
        metrics.contentLeft;
    }
    if (goog.isNumber(xyRatio.y)) {
        this.scrollY = -metrics.contentHeight * xyRatio.y -
        metrics.contentTop;
    }
    var translation = 'translate(' +
        (this.scrollX + metrics.absoluteLeft) + ',' +
        (this.scrollY + metrics.absoluteTop) + ')';
    this.getCanvas().setAttribute('transform', translation);
    this.getBubbleCanvas().setAttribute('transform',
        translation);
};

```

```
//TODO
Blockly.MiniWorkspace.prototype.renderWorkspace = function (folder, anchorX, anchorY)
     {
    this.createDom();

    Blockly.ConnectionDB.init(this);
    this.block_.expandedFolder_ = true;
    this.workspace_ = folder.workspace;
    this.shape_ = folder.svg_.svgPath_;
    var canvas = Blockly.mainWorkspace.getCanvas();
    canvas.appendChild(this.createDom_());

    this.setAnchorLocation(anchorX, anchorY);
    //Set MW  Size
    try {
        var bBox = /** @type {SVGLocatable} */ (this.svgBlockCanvas_).getBBox();
    } catch (e) {
        // Firefox has trouble with hidden elements (Bug 528969).
        var bBox = {height: 0, width: 0};
    }
    this.width_ = bBox.width + 2 * Blockly.Bubble.BORDER_WIDTH;
    this.height_ = bBox.height + 2 * Blockly.Bubble.BORDER_WIDTH;
    var doubleBorderWidth = 2 * Blockly.Bubble.BORDER_WIDTH;
    this.width_ = Math.max(this.width_, doubleBorderWidth + 45);
    this.height_ = Math.max(this.height_, 30 + Blockly.BlockSvg.FIELD_HEIGHT);
    this.svgGroupBack_.setAttribute('width',this.width_);
    this.svgGroupBack_.setAttribute('height',this.height_+20);
    this.svgGroupBack_.setAttribute('transform','translate(-5,-5)');
    this.svgGroup_.setAttribute('width',this.width_);
    this.svgTitle_.setAttribute('transform','translate(10,'+(this.height_+5)+')');


    Blockly.fireUiEvent(this.svgGroup_,'resize');

    this.positionMiniWorkspace_ ();
    this.rendered_ = true;
    this.scrollbar = new Blockly.ScrollbarPair(this);
    this.scrollbar.resize();

    if (this.xml) {
        this.clear();
        Blockly.Xml.domToWorkspace(this, this.xml);
    }

    this.render();

    if (!Blockly.readOnly) {
        Blockly.bindEvent_(this.svgGroupBack_, 'mousedown', this,
            this.miniWorkspaceMouseDown_);
    }
};

//TODO
Blockly.MiniWorkspace.prototype.disposeWorkspace = function () {
    for (var i = 1; i < 5; i++) {
        console.log(i+" "+this.connectionDBList[i].length);
    }

    Blockly.MiniWorkspace.unbindDragEvents_();
    // Dispose of and unlink the bubble.
    goog.dom.removeNode(this.svgGroup_);
    this.svgGroup_ = null;
    this.svgBlockCanvas_ = null;
    this.svgBubbleCanvas_ = null;
    this.svgGroupBack_ = null;
    this.workspace_ = null;
    this.content_ = null;
    this.shape_ = null;
    this.block_.expandedFolder_ = false;

    for (var t = 0, block; block = this.topBlocks_[t]; t++) {
        block.rendered = false;
    }
};

//MiniWorkspace cannot be resized - this can change in the future
Blockly.MiniWorkspace.prototype.createDom_ = function () {
    this.svgGroup_ = Blockly.createSvgElement('g', {}, null);
    var svgGroupEmboss = Blockly.createSvgElement('g',
        {'filter': 'url(#blocklyEmboss)'}, this.svgGroup_);

    this.svgBlockCanvasOuter_ = Blockly.createSvgElement('svg', {'height': '70%', 'width': '40%'}, this.svgGroup_);

    this.svgBlockCanvas_ = Blockly.createSvgElement('g', {}, this.svgBlockCanvasOuter_);
    Blockly.bindEvent_(this.svgBlockCanvas_, 'mousedown', this.svgBlockCanvas_,
        function(e) {
            e.preventDefault();
            e.stopPropagation();
        });

    Blockly.createSvgElement('rect',
        {'class': 'blocklyFolderBackground',
            'height': '100%', 'width': '100%'}, this.svgBlockCanvas_);

    this.svgBubbleCanvas_ = Blockly.createSvgElement('g', {'height': '100%', 'width': '100%'}, this.svgGroup_);
    this.svgGroupBack_ = Blockly.createSvgElement('rect',
        {'class': 'blocklyDraggable', 'x': 0, 'y': 0,
            'rx': Blockly.Bubble.BORDER_WIDTH, 'ry': Blockly.Bubble.BORDER_WIDTH},
        svgGroupEmboss);
    Blockly.createSvgElement('rect',
        {'class':'blocklyMutatorBackground',
            'height': '70%', 'width': '40%'}, svgGroupEmboss);
    this.svgTitle_ = Blockly.createSvgElement('text',{
        'class':'blocklyText'},this.svgGroup_);
    this.svgTitle_.innerHTML="Folder"+this.block_.id;
    this.resizeGroup_ = null;
    //this.svgBlockCanvas_.appendChild(content);

    //this.svgGroup_.appendChild(content);

    return this.svgGroup_;
};

Blockly.MiniWorkspace.prototype.addTopBlock = function(block) {
    block.workspace == this;
    block.isInFolder = true;
    this.topBlocks_.push(block);
    if (Blockly.Realtime.isEnabled() && this == Blockly.mainWorkspace) {
        Blockly.Realtime.addTopBlock(block);
    }
    this.fireChangeEvent();
};

Blockly.MiniWorkspace.prototype.setAnchorLocation = function (x,y) {
    this.anchorX_ = x;
    this.anchorY_ = y;
    if (this.rendered_) {
        this.positionMiniWorkspace_();
    }
};

Blockly.MiniWorkspace.prototype.positionMiniWorkspace_ = function () {
    var left;
    if (Blockly.RTL) {
        left = this.anchorX_ - this.relativeLeft_ - this.width_;
    } else {
        left = this.anchorX_ + this.relativeLeft_;
    }
    var top = this.relativeTop_ + this.anchorY_;
    this.svgGroup_.setAttribute('transform',
        'translate(' + left + ', ' + top + ')');
};

Blockly.MiniWorkspace.prototype.miniWorkspaceMouseDown_ = function (e) {
    this.promote_();
    Blockly.MiniWorkspace.unbindDragEvents_();
    if (Blockly.isRightButton(e)) {
        // Right-click.
        return;
    } else if (Blockly.isTargetInput_(e)) {
        // When focused on an HTML text input widget, don't trap any events.
        return;
    }
    // Left-click (or middle click)
    Blockly.setCursorHand_(true);
    // Record the starting offset between the current location and the mouse.
    if (Blockly.RTL) {
        this.dragDeltaX =  this.relativeLeft_ + e.clientX;
    } else {
        this.dragDeltaX = this.relativeLeft_ - e.clientX;
    }
    this.dragDeltaY = this.relativeTop_ - e.clientY;

    Blockly.MiniWorkspace.onMouseUpWrapper_ = Blockly.bindEvent_(document,
        'mouseup', this, Blockly.MiniWorkspace.unbindDragEvents_);
    Blockly.MiniWorkspace.onMouseMoveWrapper_ = Blockly.bindEvent_(document,
        'mousemove', this, this.MiniWorkspaceMouseMove_);
    Blockly.hideChaff();
    // This event has been handled.  No need to bubble up to the document.
    e.stopPropagation();
};

Blockly.MiniWorkspace.unbindDragEvents_ = function() {
    if (Blockly.MiniWorkspace.onMouseUpWrapper_) {
        Blockly.unbindEvent_(Blockly.MiniWorkspace.onMouseUpWrapper_);
        Blockly.MiniWorkspace.onMouseUpWrapper_ = null;
    }
    if (Blockly.MiniWorkspace.onMouseMoveWrapper_) {
        Blockly.unbindEvent_(Blockly.MiniWorkspace.onMouseMoveWrapper_);
        Blockly.MiniWorkspace.onMouseMoveWrapper_ = null;
    }
};

Blockly.MiniWorkspace.prototype.MiniWorkspaceMouseMove_ = function(e) {
    this.autoLayout_ = false;
    if (Blockly.RTL) {
        this.relativeLeft_ = this.dragDeltaX - e.clientX;
    } else {
        this.relativeLeft_ = this.dragDeltaX + e.clientX;
    }
    this.relativeTop_ = this.dragDeltaY + e.clientY;
    this.positionMiniWorkspace_();
};

Blockly.MiniWorkspace.prototype.promote_ = function() {
    var svgGroup = this.svgGroup_.parentNode;
    svgGroup.appendChild(this.svgGroup_);
    this.block_.promote();
};

Blockly.MiniWorkspace.prototype.highlight_ = function() {
    Blockly.addClass_(/** @type {!Element} */ (this.svgGroupBack_),
        'blocklySelectedFolder');
};

Blockly.MiniWorkspace.prototype.unhighlight_ = function() {
    Blockly.removeClass_(/** @type {!Element} */ (this.svgGroupBack_),
        'blocklySelectedFolder');
};
```

Listing E.1: appinventor/blocklyeditor/src/miniworkspace.js

Appendix F

Yail.js

Path to file: appinventor/lib/blockly/src/core/yail.js. Functions with changes are listed below:

```
Blockly.Yail.getDebuggingYail = function() {
  var code = [];
  var componentMap = Blockly.Component.buildComponentMap([], [], false, false);

  var globalBlocks = componentMap.globals;
  for (var i = 0, block; block = globalBlocks[i]; i++) {
    code.push(Blockly.Yail.blockToCode(block));
  }

  var blocks = Blockly.mainWorkspace.getTopBlocks(true);
    //[Shirley 3/21] post-process of topBlocks

    var blocks2 = [];
    for (var x = 0, block; block = blocks[x]; x++) {
        if (block.category == "Folders") {
            blocks2 = blocks2.concat(block.miniworkspace.topBlocks_);
        } else {
            blocks2 = blocks2.concat(block);
        }
    }
    blocks = blocks2;
    //[Shirley 3/21] end

  for (var x = 0, block; block = blocks[x]; x++) {

    // generate Yail for each top-level language block
    if (!block.category) {
      continue;
    }
    code.push(Blockly.Yail.blockToCode(block));
  }
  return code.join('\n\n');
};
```

Listing F.1: Blockly.Yail.getDebuggingYail

Appendix G

Block.js

Path to file: appinventor/lib/blockly/src/core/block.js. Functions with changes are listed below:

```
Blockly.Block.prototype.dispose = function(healStack, animate,
                                           dontRemoveFromWorkspace) {
  if (this.type == "folder") {
    this.miniworkspace.dispose();
  }

  // Switch off rerendering.
  this.rendered = false;
  this.unplug(healStack);

  if (animate && this.svg_) {
    this.svg_.disposeUiEffect();
  }

  // This block is now at the top of the workspace.
  // Remove this block from the workspace's list of top-most blocks.
  if (this.workspace && !dontRemoveFromWorkspace) {
    this.workspace.removeTopBlock(this);
    this.workspace = null;
  }

  // Just deleting this block from the DOM would result in a memory leak as
  // well as corruption of the connection database.  Therefore we must
  // methodically step through the blocks and carefully disassemble them.

  if (Blockly.selected == this) {
    Blockly.selected = null;
    // If there's a drag in-progress, unlink the mouse events.
    Blockly.terminateDrag_();
  }

  // If this block has a context menu open, close it.
  if (Blockly.ContextMenu.currentBlock == this) {
    Blockly.ContextMenu.hide();
  }

  // First, dispose of all my children.
  for (var x = this.childBlocks_.length - 1; x >= 0; x--) {
    this.childBlocks_[x].dispose(false);
  }
  // Then dispose of myself.
  var icons = this.getIcons();
  for (var x = 0; x < icons.length; x++) {
    icons[x].dispose();
  }
  if (this.errorIcon) {
    this.errorIcon.dispose();
  }

  // Dispose of all inputs and their fields.
  for (var x = 0, input; input = this.inputList[x]; x++) {
    input.dispose();
  }
  this.inputList = [];
  // Dispose of any remaining connections (next/previous/output).
  var connections = this.getConnections_(true);
  for (var x = 0; x < connections.length; x++) {
    var connection = connections[x];
    if (connection.targetConnection) {
      connection.disconnect();
    }
    connections[x].dispose();
  }
  // Dispose of the SVG and break circular references.
  if (this.svg_) {
    this.svg_.dispose();
    this.svg_ = null;
  }
  // Remove from Realtime set of blocks.
  if (Blockly.Realtime.isEnabled() && !Blockly.Realtime.withinSync) {
    Blockly.Realtime.removeBlock(this);
  }
  // Remove any associated errors or warnings.
  Blockly.WarningHandler.checkDisposedBlock.call(this);
};
```

Listing G.1: Blockly.Block.prototype.dispose

```
Blockly.Block.prototype.onMouseDown_ = function(e) {
  if (this.isInFlyout) {
    return;
  }
  // Update Blockly's knowledge of its own location.
  Blockly.svgResize();
  Blockly.terminateDrag_();

  this.select();
  Blockly.hideChaff();
  if (Blockly.isRightButton(e)) {
    // Right-click.
    this.showContextMenu_(e);
  } else if (!this.isMovable()) {
    // Allow unmovable blocks to be selected and context menued, but not
    // dragged.  Let this event bubble up to document, so the workspace may be
    // dragged instead.
    return;
  } else {
    // Left-click (or middle click)
    Blockly.removeAllRanges();
    Blockly.setCursorHand_(true);
    // Look up the current translation and record it.
    var xy = this.getRelativeToSurfaceXY();
    this.startDragX = xy.x;
    this.startDragY = xy.y;
    // Record the current mouse position.
    this.startDragMouseX = e.clientX;
    this.startDragMouseY = e.clientY;
    Blockly.Block.dragMode_ = 1;
    Blockly.Block.onMouseUpWrapper_ = Blockly.bindEvent_(document,
        'mouseup', this, this.onMouseUp_);
    Blockly.Block.onMouseMoveWrapper_ = Blockly.bindEvent_(document,
        'mousemove', this, this.onMouseMove_);
    // Build a list of bubbles that need to be moved and where they started.
    this.draggedBubbles_ = [];
    var descendants = this.getDescendants();
    for (var x = 0, descendant; descendant = descendants[x]; x++) {
      var icons = descendant.getIcons();
      for (var y = 0; y < icons.length; y++) {
        var data = icons[y].getIconLocation();
        data.bubble = icons[y];
        this.draggedBubbles_.push(data);
      }
      if (descendant.errorIcon) {
        var data = descendant.errorIcon.getIconLocation();
        data.bubble = descendant.errorIcon;
        this.draggedBubbles_.push(data);
      }
    }
  }
  // This event has been handled.  No need to bubble up to the document.
  e.stopPropagation();
};
```

```
Blockly.Block.prototype.onMouseUp_ = function(e) {
  var start = new Date().getTime();
  Blockly.Instrument.initializeStats("onMouseUp");
  var this_ = this;
  Blockly.resetWorkspaceArrangements();
  Blockly.doCommand(function() {
    Blockly.terminateDrag_();

    if (Blockly.selectedFolder_) {
      Blockly.selectedFolder_.miniworkspace.moveBlock(this_);
    }


    if (Blockly.selected && Blockly.highlightedConnection_) {
      // Connect two blocks together.
      Blockly.localConnection_.connect(Blockly.highlightedConnection_);
      if (this_.svg_) {
        // Trigger a connection animation.
        // Determine which connection is inferior (lower in the source stack).
        var inferiorConnection;
        if (Blockly.localConnection_.isSuperior()) {
          inferiorConnection = Blockly.highlightedConnection_;
        } else {
          inferiorConnection = Blockly.localConnection_;
        }
        inferiorConnection.sourceBlock_.svg_.connectionUiEffect();
      }
      if (this_.workspace.trashcan && this_.workspace.trashcan.isOpen) {
        // Don't throw an object in the trash can if it just got connected.
        this_.workspace.trashcan.close();
      }
    } else if (this_.workspace.trashcan && this_.workspace.trashcan.isOpen) {
      var trashcan = this_.workspace.trashcan;
      goog.Timer.callOnce(trashcan.close, 100, trashcan);
      if (Blockly.selected.confirmDeletion()) {
        Blockly.selected.dispose(false, true);
      }
      // Dropping a block on the trash can will usually cause the workspace to
      // resize to contain the newly positioned block.  Force a second resize
      // now that the block has been deleted.
      Blockly.fireUiEvent(window, 'resize');
    }

    if (Blockly.highlightedConnection_) {
      Blockly.highlightedConnection_.unhighlight();
      Blockly.highlightedConnection_ = null;
    }

    if (Blockly.selectedFolder_) {
      Blockly.selectedFolder_.miniworkspace.unhighlight_();
      Blockly.selectedFolder_ = null;
    }

  });
  if (! Blockly.Instrument.avoidRenderWorkspaceInMouseUp) {
    // [lyn, 04/01/14] rendering a workspace takes a *long* time and is *not*
        necessary!
    // This is the key source of the laggy drag problem. Remove it!
    Blockly.mainWorkspace.render();
  }
  Blockly.WarningHandler.checkAllBlocksForWarningsAndErrors();
  var stop = new Date().getTime();
  var timeDiff = stop - start;
  Blockly.Instrument.stats.totalTime = timeDiff;
  Blockly.Instrument.displayStats("onMouseUp");
};
```

Listing G.2: Blockly.Block.prototype.onMouseUp_

```
Blockly.Block.prototype.onMouseMove_ = function(e) {
  var this_ = this;
  Blockly.doCommand(function() {
    if (e.type == 'mousemove' && e.clientX <= 1 && e.clientY == 0 &&
        e.button == 0) {
      /* HACK:
       Safari Mobile 6.0 and Chrome for Android 18.0 fire rogue mousemove events
       on certain touch actions. Ignore events with these signatures.
       This may result in a one-pixel blind spot in other browsers,
       but this shouldn't be noticable. */
      e.stopPropagation();
      return;
    }
    Blockly.removeAllRanges();
    var dx = e.clientX - this_.startDragMouseX;
    var dy = e.clientY - this_.startDragMouseY;
    if (Blockly.Block.dragMode_ == 1) {
      // Still dragging within the sticky DRAG_RADIUS.
      var dr = Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2));
      if (dr > Blockly.DRAG_RADIUS) {
        // Switch to unrestricted dragging.
        Blockly.Block.dragMode_ = 2;
        // Push this block to the very top of the stack.
        this_.setParent(null);
        this_.setDragging_(true);
      }
    }
    // [Shirley 4/11] - everytime a block is clicked, it is put in the mainWorkspace
    if (this_.workspace.isMW) {
      var transformMatrix = Blockly.mainWorkspace.moveOutOfFolder(this_);
      this_.startDragX += transformMatrix[0];
      this_.startDragY += transformMatrix[1];
    }
    if (Blockly.Block.dragMode_ == 2) {
      // Unrestricted dragging.
      //  console.log("drag " + this_.startDragX+ " "+ this_.startDragY+ " "+dx+" "+dy);
      var x = this_.startDragX + dx;
      var y = this_.startDragY + dy;
        //console.log("drag2 "+x+" "+y);
      this_.svg_.getRootElement().setAttribute('transform',
          'translate(' + x + ', ' + y + ')');
      // Drag all the nested bubbles.
      for (var i = 0; i < this_.draggedBubbles_.length; i++) {
        var commentData = this_.draggedBubbles_[i];
        commentData.bubble.setIconLocation(commentData.x + dx,
            commentData.y + dy);
      }

      //find the folder the block is over
      var overFolder = null;
      for (var i = 0; i < Blockly.ALL_FOLDERS.length; i++) {
        if (this_ != Blockly.ALL_FOLDERS[i] &&
            Blockly.ALL_FOLDERS[i].isOverFolder(e)) {
          overFolder = Blockly.ALL_FOLDERS[i];
          break;
        }
      }
      //remove highlighting if necessary
      if (Blockly.selectedFolder_ &&
          Blockly.selectedFolder_ != overFolder) {
        Blockly.selectedFolder_.miniworkspace.unhighlight_();
        Blockly.selectedFolder_ = null;
      }
      //add highlighting if necessary
      if (overFolder && overFolder != Blockly.selectedFolder_) {
        Blockly.selectedFolder_ = overFolder;
        Blockly.selectedFolder_.miniworkspace.highlight_();
      }

      // Check to see if any of this block's connections are within range of
      // another block's connection.
      var myConnections = this_.getConnections_(false);
      var closestConnection = null;
      var localConnection = null;
      var radiusConnection = Blockly.SNAP_RADIUS;
      for (var i = 0; i < myConnections.length; i++) {
        var myConnection = myConnections[i];
        var neighbour = myConnection.closest(radiusConnection, dx, dy, Blockly.
            selectedFolder_);
        if (neighbour.connection) {
          closestConnection = neighbour.connection;
          localConnection = myConnection;
          radiusConnection = neighbour.radius;
        }
      }

      // Remove connection highlighting if needed.
      if (Blockly.highlightedConnection_ &&
          Blockly.highlightedConnection_ != closestConnection) {
        Blockly.highlightedConnection_.unhighlight();
        Blockly.highlightedConnection_ = null;
        Blockly.localConnection_ = null;
      }

      // Add connection highlighting if needed.
      if (closestConnection &&
          closestConnection != Blockly.highlightedConnection_) {
        closestConnection.highlight();
        Blockly.highlightedConnection_ = closestConnection;
        Blockly.localConnection_ = localConnection;
      }


      // Flip the trash can lid if needed.
      if (this_.workspace.trashcan && this_.isDeletable()) {
        this_.workspace.trashcan.onMouseMove(e);
      }
    }
    // This event has been handled.  No need to bubble up to the document.
    e.stopPropagation();
  });
```

```
};
```

Listing G.3: Blockly.Block.prototype.onMouseMove_

Appendix H

Connection.js

Path to file: appinventor/lib/blockly/src/core/connection.js. Functions with changes are listed below:

```
Blockly.Connection.prototype.moveTo = function(x, y) {
  // Remove it from its old location in the database (if already present)
  if (this.inDB_) {
    this.dbList_[this.type].removeConnection_(this);
  }
  this.x_ = x;
  this.y_ = y;
  // Insert it into its new location in the database.
    if (!this.dbList_) {
        this.dbList_ = this.sourceBlock_.workspace.workspace_.connectionDBList;
    }
  this.dbList_[this.type].addConnection_(this);
};
```

Listing H.1: Blockly.Connection.prototype.moveTo

```
Blockly.Connection.prototype.closest = function(maxLimit, dx, dy, folder) {
  if (this.targetConnection) {
    // Don't offer to connect to a connection that's already connected.
    return {connection: null, radius: maxLimit};
  }
  // Determine the opposite type of connection.
  var oppositeType = Blockly.OPPOSITE_TYPE[this.type];
  var db = this.dbList_[oppositeType];
  var folderdx = 0;
  var folderdy = 0;

  if (folder) {
    db = folder.miniworkspace.connectionDBList[oppositeType];
    var folderOrigin = Blockly.getRelativeXY_(folder.miniworkspace.svgGroup_);
    var translate_ = folder.miniworkspace.getTranslate();
    folderdx = folderOrigin.x + parseInt(translate_[0]);
    folderdy = folderOrigin.y + parseInt(translate_[1]);
  }

  // Since this connection is probably being dragged, add the delta.
  var currentX = this.x_ + dx - folderdx;
  var currentY = this.y_ + dy - folderdy;

  // Binary search to find the closest y location.
  var pointerMin = 0;
  var pointerMax = db.length - 2;
  var pointerMid = pointerMax;
  while (pointerMin < pointerMid) {
    if (db[pointerMid].y_ < currentY) {
      pointerMin = pointerMid;
    } else {
      pointerMax = pointerMid;
    }
    pointerMid = Math.floor((pointerMin + pointerMax) / 2);
  }

  // Walk forward and back on the y axis looking for the closest x,y point.
  pointerMin = pointerMid;
  pointerMax = pointerMid;
  var closestConnection = null;
  var sourceBlock = this.sourceBlock_;
  var thisConnection = this;
  if (db.length) {
    while (pointerMin >= 0 && checkConnection_(pointerMin)) {
      pointerMin--;
    }
    do {
      pointerMax++;
    } while (pointerMax < db.length && checkConnection_(pointerMax));
  }
```

```

/**
 * Computes if the current connection is within the allowed radius of another
 * connection.
 * This function is a closure and has access to outside variables.
 * @param {number} yIndex The other connection's index in the database.
 * @return {boolean} True if the search needs to continue: either the current
 *     connection's vertical distance from the other connection is less than
 *     the allowed radius, or if the connection is not compatible.
 */
function checkConnection_(yIndex) {
  var connection = db[yIndex];
  if (connection.type == Blockly.OUTPUT_VALUE ||
      connection.type == Blockly.PREVIOUS_STATEMENT) {
    // Don't offer to connect an already connected left (male) value plug to
    // an available right (female) value plug.  Don't offer to connect the
    // bottom of a statement block to one that's already connected.
    if (connection.targetConnection) {
      return true;
    }
  }
  // Offering to connect the top of a statement block to an already connected
  // connection is ok, we'll just insert it into the stack.

  // Offering to connect the left (male) of a value block to an already
  // connected value pair is ok, we'll splice it in.
  // However, don't offer to splice into an unmovable block.
  if (connection.type == Blockly.INPUT_VALUE &&
      connection.targetConnection &&
      !connection.targetBlock().isMovable()) {
    return true;
  }

  // Do type checking.
  if (!thisConnection.checkType_(connection)) {
    return true;
  }

  // Don't let blocks try to connect to themselves or ones they nest.
  var targetSourceBlock = connection.sourceBlock_;
  do {
      if (sourceBlock == targetSourceBlock) {
        return true;
      }
      targetSourceBlock = targetSourceBlock.getParent();
    } while (targetSourceBlock);

    var dx = currentX - db[yIndex].x_;
    var dy = currentY - db[yIndex].y_;
    var r = Math.sqrt(dx * dx + dy * dy);
    if (r <= maxLimit) {
      closestConnection = db[yIndex];
      maxLimit = r;
    }
    return dy < maxLimit;
  }
  return {connection: closestConnection, radius: maxLimit};
};
```

Listing H.2: Blockly.Connection.prototype.closest

Appendix I

Workspace.js

Path to file: appinventor/lib/blockly/src/core/workspace.js.

```
Blockly.Workspace.prototype.moveBlock = function(block) {

  this.moveIntoFolder(block);
  this.moveChild(block);
};

//newWorkspace.moveIntoFolder(block)
Blockly.Workspace.prototype.moveIntoFolder = function (block) {
  // The oldWorkspace will always be the mainWorkspace
  var oldWorkspace = Blockly.mainWorkspace;
  // newWorkspace will always be this
  var newWorkspace = this;

  // Move the Block into the right place in the folder
  var blockRelativeToMWXY = block.getRelativeToSurfaceXY();
  var miniWorkspaceOrigin = Blockly.getRelativeXY_(this.svgGroup_);
  Blockly.mainWorkspace.removeTopBlock(block);
  this.addTopBlock(block);
  //surgically removes all svg associated with block from old workspace canvas
  var svgGroup = goog.dom.removeNode(block.svg_.svgGroup_);
  block.workspace = this;
  this.getCanvas().appendChild(svgGroup);

  var translate_ = this.getTranslate();
  var dx = -1 * (miniWorkspaceOrigin.x + parseInt(translate_[0]));
  var dy = -1 * (miniWorkspaceOrigin.y + parseInt(translate_[1]));
  var x = blockRelativeToMWXY.x + dx;
  var y = blockRelativeToMWXY.y + dy;
  block.svg_.getRootElement().setAttribute('transform',
      'translate(' + x + ', ' + y + ')');

  // remove, change x & y, add
  if (block.outputConnection) {
    changeConnection(block.outputConnection);
  }
  if (block.nextConnection) {
    changeConnection(block.nextConnection);
  }
  if (block.previousConnection) {
    changeConnection(block.previousConnection);
  }
  if (block.inputList) {
    for (var i = 0; i < block.inputList.length; i++) {
      var c = block.inputList[i];
      if (c.connection) {
        changeConnection(c.connection);
      }
    }
  }

  function changeConnection (connect) {
    oldWorkspace.connectionDBList[connect.type].removeConnection_(connect);
    connect.x_ += dx;
    connect.y_ += dy;
    newWorkspace.connectionDBList[connect.type].addConnection_(connect);
    if (connect.targetConnection) {
      var tconnect = connect.targetConnection;
      oldWorkspace.connectionDBList[tconnect.type].removeConnection_(tconnect);
      tconnect.x_ += dx;
      tconnect.y_ += dy;
      newWorkspace.connectionDBList[tconnect.type].addConnection_(tconnect);
      tconnect.dbList_ = newWorkspace.connectionDBList;
    }
    connect.dbList_ = newWorkspace.connectionDBList;
  }

};
```

```

//newWorkspace.moveOutOfFolder(block)
Blockly.Workspace.prototype.moveOutOfFolder = function (block) {
  // this is used everytime a block is clicked - if it's in main, don't move it
  if (block.workspace == Blockly.mainWorkspace) {
      return;
  }

  //Move block into the right place in the main workspace
  var oldWorkspace = block.workspace;
  var newWorkspace = this;
  var blockRelativeToWXY = block.getRelativeToSurfaceXY();
  var miniWorkspaceOrigin = Blockly.getRelativeXY_(oldWorkspace.svgGroup_);
  oldWorkspace.removeTopBlock(block);
  newWorkspace.addTopBlock(block);
  //surgically removes all svg associated with block from old workspace canvas
  var svgGroup = goog.dom.removeNode(block.svg_.svgGroup_);
  block.workspace = newWorkspace;
  newWorkspace.getCanvas().appendChild(svgGroup);

  var translate_ = oldWorkspace.getTranslate();
  var dx = miniWorkspaceOrigin.x + parseInt(translate_[0]);
  var dy = miniWorkspaceOrigin.y + parseInt(translate_[1]);
  var x = blockRelativeToWXY.x + dx;
  var y = blockRelativeToWXY.y + dy;
  block.svg_.getRootElement().setAttribute('transform',
      'translate(' + x + ', ' + y + ')');
  block.isInFolder = false;

  // Change the old workspace and new workspace's connectionDBList
  if (block.outputConnection) {
    changeConnection(block.outputConnection);
  }
  if (block.nextConnection) {
    changeConnection(block.nextConnection);
  }
  if (block.previousConnection) {
    changeConnection(block.previousConnection);
  }
  if (block.inputList) {
    for (var i = 0; i < block.inputList.length; i++) {
      var c = block.inputList[i];
      if (c.connection) {
        changeConnection(c.connection);
      }
    }
  }

  function changeConnection (connect) {
    oldWorkspace.connectionDBList[connect.type].removeConnection_(connect);
    connect.x_ += dx;
    connect.y_ += dy;
    newWorkspace.connectionDBList[connect.type].addConnection_(connect);
    if (connect.targetConnection) {
      var tconnect = connect.targetConnection;
      oldWorkspace.connectionDBList[tconnect.type].removeConnection_(tconnect);
      tconnect.x_ += dx;
      tconnect.y_ += dy;
      newWorkspace.connectionDBList[tconnect.type].addConnection_(tconnect);
      tconnect.dbList_ = newWorkspace.connectionDBList;
    }
    connect.dbList_ = newWorkspace.connectionDBList;
  }

  newWorkspace.moveChild(block);

  return [dx,dy];

};

Blockly.Workspace.prototype.moveChild = function(block){
  for (var cb = 0; cb < block.childBlocks_.length; cb++) {
    var childBlock = block.childBlocks_[cb];
    this.moveChild(childBlock);
    childBlock.workspace = this;
  }
}

Blockly.Workspace.prototype.getTranslate = function () {
    var translate = this.getCanvas().getAttribute("transform");
    translate = translate.split("(")[1].split(")")[0];
    return translate.split(",");
```

```
};
```

Listing I.1: New functions added to workspace.js

Appendix J

XML.js

Path to file: appinventor/lib/blockly/src/core/xml.js. Functions with changes are listed below:

```
Blockly.Xml.workspaceToDom = function(workspace) {
  var width;  // Not used in LTR.
  if (Blockly.RTL) {
    width = workspace.getMetrics().viewWidth;
  }
  var xml = goog.dom.createDom('xml');
  var blocks = workspace.getTopBlocks(true);
  for (var i = 0, block; block = blocks[i]; i++) {
    var element = Blockly.Xml.blockToDom_(block);
      if (block.type == "folder") {
          var folder = Blockly.Xml.workspaceToDom(block.miniworkspace);
          for (var x = 0, b; b = folder.childNodes[x];){
              element.appendChild(b);
          }
      }
    var xy = block.getRelativeToSurfaceXY();
    element.setAttribute('x', Blockly.RTL ? width - xy.x : xy.x);
    element.setAttribute('y', xy.y);
    xml.appendChild(element);
  }
  return xml;
};
```

Listing J.1: Blockly.Xml.workspaceToDom

```
Blockly.Xml.domToWorkspace = function(workspace, xml) {
  Blockly.Instrument.timer (
      function () {
        var width;  // Not used in LTR.
        if (Blockly.RTL) {
          width = workspace.getMetrics().viewWidth;
        }
// The commented line below was replaced because it would reference beyond
// the end of the childNodes pseudo-array. In Chrome this is fine because
// the value returned is "undefined" which counts as false. However when
// using phantomjs (unit test) you wind up fetching memory garbage (!!)
//
//        for (var x = 0, xmlChild; xmlChild = xml.childNodes[x]; x++) {
        var xmlChild;
        for (var x = 0; x < xml.childNodes.length; x++) {
          xmlChild = xml.childNodes[x];
          if (xmlChild.nodeName.toLowerCase() == 'block') {
            var block = Blockly.Xml.domToBlock(workspace, xmlChild);
              if (block.type == "folder") {
                  var folderXML = goog.dom.createDom('xml');
                  while(xmlChild.children.length > 0) {
                      folderXML.appendChild(xmlChild.children[0]);
                  }
                  block.miniworkspace.xml = folderXML;
              }
            var blockX = parseInt(xmlChild.getAttribute('x'), 10);
            var blockY = parseInt(xmlChild.getAttribute('y'), 10);
            if (!isNaN(blockX) && !isNaN(blockY)) {
              block.moveBy(Blockly.RTL ? width - blockX : blockX, blockY);
            }
          }
        }
      },
      function (result, timeDiff) {
        Blockly.Instrument.stats.domToWorkspaceCalls++;
        Blockly.Instrument.stats.domToWorkspaceTime = timeDiff;
      }
  );
};
```

Listing J.2: Blockly.Xml.domToWorkspace

```
Blockly.Xml.domToBlockInner = function(workspace, xmlBlock, opt_reuseBlock) {
  Blockly.Instrument.stats.domToBlockInnerCalls++;
```

```
var block = null;
var prototypeName = xmlBlock.getAttribute('type');
if (!prototypeName) {
  throw 'Block type unspecified: \n' + xmlBlock.outerHTML;
}
var id = xmlBlock.getAttribute('id');
if (opt_reuseBlock && id) {
  block = Blockly.Block.getById(id, workspace);
  // TODO: The following is for debugging.  It should never actually happen.
  if (!block) {
    throw 'Couldn\'t get Block with id: ' + id;
  }
  var parentBlock = block.getParent();
  // If we've already filled this block then we will dispose of it and then
  // re-fill it.
  if (block.workspace) {
    block.dispose(true, false, true);
  }
  block.fill(workspace, prototypeName);
  block.parent_ = parentBlock;
} else {
    if (prototypeName == "folder") {
      //here block is actually a Blockly.Folder() instance
      block = Blockly.Folder.obtain(workspace,prototypeName);
    } else {
      block = Blockly.Block.obtain(workspace, prototypeName);
    }
}
if (!block.svg_) {
  block.initSvg();
}

var inline = xmlBlock.getAttribute('inline');
if (inline) {
  block.setInputsInline(inline == 'true');
}
var disabled = xmlBlock.getAttribute('disabled');
if (disabled) {
  block.setDisabled(disabled == 'true');
}
var deletable = xmlBlock.getAttribute('deletable');
```

```
  if (deletable) {
    block.setDeletable(deletable == 'true');
  }
  var movable = xmlBlock.getAttribute('movable');
  if (movable) {
    block.setMovable(movable == 'true');
  }
  var editable = xmlBlock.getAttribute('editable');
  if (editable) {
    block.setEditable(editable == 'true');
  }

  var blockChild = null;
  for (var x = 0, xmlChild; xmlChild = xmlBlock.childNodes[x]; x++) {
    if (xmlChild.nodeType == 3 && xmlChild.data.match(/^\s*$/)) {
      // Extra whitespace between tags does not concern us.
      continue;
    }
    var input;

    // Find the first 'real' grandchild node (that isn't whitespace).
    var firstRealGrandchild = null;
    for (var y = 0, grandchildNode; grandchildNode = xmlChild.childNodes[y];
         y++) {
      if (grandchildNode.nodeType != 3 || !grandchildNode.data.match(/^\s*$/)) {
        firstRealGrandchild = grandchildNode;
      }
    }

    var name = xmlChild.getAttribute('name');
    switch (xmlChild.nodeName.toLowerCase()) {
      case 'mutation':
        // Custom data for an advanced block.
        if (block.domToMutation) {
          block.domToMutation(xmlChild);
        }
        break;
      case 'comment':
        block.setCommentText(xmlChild.textContent);
        var visible = xmlChild.getAttribute('pinned');
        if (visible) {
          // Give the renderer a millisecond to render and position the block
          // before positioning the comment bubble.
          setTimeout(function() {
            block.comment.setVisible(visible == 'true');
          }, 1);
        }
        var bubbleW = parseInt(xmlChild.getAttribute('w'), 10);
        var bubbleH = parseInt(xmlChild.getAttribute('h'), 10);
        if (!isNaN(bubbleW) && !isNaN(bubbleH)) {
          block.comment.setBubbleSize(bubbleW, bubbleH);
        }
        break;
      case 'title':
        // Titles were renamed to field in December 2013.
        // Fall through.
      case 'field':
        block.setFieldValue(xmlChild.textContent, name);
        break;
      case 'value':
      case 'statement':
        input = block.getInput(name);
        if (!input) {
          throw 'Input ' + name + ' does not exist in block ' + prototypeName;
        }
        if (firstRealGrandchild &&
            firstRealGrandchild.nodeName.toLowerCase() == 'block') {
          blockChild = Blockly.Xml.domToBlockInner(workspace, firstRealGrandchild,
              opt_reuseBlock);
          if (blockChild.outputConnection) {
            input.connection.connect(blockChild.outputConnection);
          } else if (blockChild.previousConnection) {
            input.connection.connect(blockChild.previousConnection);
          } else {
            throw 'Child block does not have output or previous statement.';
          }
        }
        break;
      case 'next':
        if (firstRealGrandchild &&
            firstRealGrandchild.nodeName.toLowerCase() == 'block') {
          if (!block.nextConnection) {
            throw 'Next statement does not exist.';
          } else if (block.nextConnection.targetConnection) {
            // This could happen if there is more than one XML 'next' tag.
            throw 'Next statement is already connected.';
          }
          blockChild = Blockly.Xml.domToBlockInner(workspace, firstRealGrandchild,
              opt_reuseBlock);
          if (!blockChild.previousConnection) {
            throw 'Next block does not have previous statement.';
          }
          block.nextConnection.connect(blockChild.previousConnection);
        }
        break;
      default:
        // Unknown tag; ignore.  Same principle as HTML parsers.
    }
  }

  // [lyn, 10/25/13] collapsing and friends need to be done *after* connections are made to sublocks.
  // Otherwise, the subblocks won't be properly processed by block.setCollapsed and friends.
  var inline = xmlBlock.getAttribute('inline');
  if (inline) {
    block.setInputsInline(inline == 'true');
  }
  var disabled = xmlBlock.getAttribute('disabled');
  if (disabled) {
    block.setDisabled(disabled == 'true');
  }
  var deletable = xmlBlock.getAttribute('deletable');
  if (deletable) {
    block.setDeletable(deletable == 'true');
  }
  var movable = xmlBlock.getAttribute('movable');
  if (movable) {
    block.setMovable(movable == 'true');
  }
  var editable = xmlBlock.getAttribute('editable');
  if (editable) {
    block.setEditable(editable == 'true');
  }

  if (! Blockly.Instrument.useRenderDown) {
    // Neil's original rendering code
    var next = block.getNextBlock();
    if (next) {
      // Next block in a stack needs to square off its corners.
      // Rendering a child will render its parent.
      next.render();
    } else {
      block.render();
    }
  }
  var collapsed = xmlBlock.getAttribute('collapsed');
  if (collapsed) {
    block.setCollapsed(collapsed == 'true');
  }
  return block;
};
```

Listing J.3: Blockly.Xml.domToBlockInner